Jül-4392

Institute for Advanced Simulation (IAS)
Jülich Supercomputing Centre (JSC)

Der hybride Parareal / SDC-Algorithmus

Selman Terzi

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Der hybride Parareal / SDC-Algorithmus

Selman Terzi

Berichte des Forschungszentrums Jülich; 4392
ISSN 0944-2952
Institute for Advanced Simulation (IAS)
Jülich Supercomputing Centre (JSC)
Jül-4392

(Master, FH Aachen, Campus Jülich, 2015)

Vollständig frei verfügbar über das Publikationsportal des Forschungszentrums Jülich (JuSER)
unter www.fz-juelich.de/zb/openaccess

Forschungszentrum Jülich GmbH
Zentralbibliothek, Verlag
52425 Jülich
Tel.: +49 2461 61-5220
Fax: +49 2461 61-6103
E-Mail: zb-publikation@fz-juelich.de
www.fz-juelich.de/zb

Zusammenfassung

Die stets ansteigende Zahl der Kerne in heutigen und kommenden Supercomputern forciert sowohl die Notwendigkeit als auch das Interesse an neuen parallelen Algorithmen. Als Ergänzung zu klassischen raumparallelen Ansätzen erlauben zeitparallele Algorithmen die Lösung von Anfangswertproblemen parallel in der Zeitdimension. Ein prominenter und umfangreich untersuchter Algorithmus in diesem Bereich ist der Parareal-Algorithmus von Lions et al. aus dem Jahre 2001. Ein wesentlicher Vorteil von Parareal ist, dass klassische Lösungsverfahren auf iterative Weise wiederverwendet werden können, um die Parallelisierung in der Zeit zu erlauben. Eine Weiterentwicklung mit dem Zweck der Verbesserung der parallelen Effizienz ist der hybride Parareal/SDC-Algorithmus aus dem Jahre 2010. Dabei wird das Fehlerkorrekturverfahren Spectral Deferred Corrections (SDC) mit Parareal so verknüpft, dass eine bessere parallele Effizienz erreicht werden kann. In dieser Arbeit werden zwei Varianten des hybriden Parareal/SDC-Algorithmus anhand einer eigenen Implementierung im PFASST++-Framework untereinander und mit dem klassischen Parareal-Algorithmus hinsichtlich der parallelen Effizienz verglichen. Die Theorie konnte dahingehend bestätigt werden, dass durch die hybriden Parareal/SDC-Varianten bessere Speedups erreicht werden können. Dieser Ansatz schlägt eine Brücke zwischen dem klassischen Parareal-Algorithmus und dem Full Approximation Scheme in Space and Time (PFASST), der im Jahre 2012 veröffentlicht wurde. Die Arbeit kann dazu beigetragen, dass die Entwicklung des Parareal-Algorithmus hin zum PFASST-Algorithmus im Rahmen eines einzigen Frameworks besser nachvollzogen werden kann.

Abstract

The ever rising number of cores in recent and upcoming supercomputers enforces both the demand for and interest in new parallel algorithms. In addition to classical space-parallel approaches, time-parallel algorithms allow the solution of initial value problems parallel in the temporal dimension. A prominent and well-examined algorithm in this field is the Parareal algorithm by Lions et al. from 2001. A major advantage of Parareal is the ability of reusing standard serial methods in an iterative fashion in order to introduce concurrency in time. An improvement of Parareal with the aim of enhancing the parallel efficiency is the hybrid Parareal/SDC algorithm from 2010. Here, the spectral deferred corrections (SDC) method is combined with Parareal in such a way that the parallel efficiency can be improved. In this thesis, two variants of the hybrid Parareal/SDC algorithm are examined on the basis of an implementation in the PFASST++-framework. The different algorithms are compared to each other and to the classical Parareal approach with respect to the parallel efficiency. The theory is confirmed in the point that the hybrid Parareal/SDC variants achieve higher speedups. This hybrid approach establishes the tie between the classical Parareal algorithm and the recently developed Parallel Full Approximation Scheme in Space and Time (PFASST). This thesis can therefore help to retrace the evolution from Parareal to the PFASST algorithm within a single framework.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

Symbol	Beschreibung
n	Zeitschrittindex
k	Iterationsindex
U_n^k	Näherung zum Zeitschritt n in der Iteration k
T_0	Startzeitpunkt
T	Endzeitpunkt
Δt	Schrittweite der zeitlichen Diskretisierung
N_p	Anzahl der für die parallele Berechnung eingesetzten Prozesse
G	Grober Propagator
F	Feiner Propagator
G_{SDC}	Grober SDC Propagator
F_{SDC}	Feiner SDC Propagator
j	Index des Quadraturknotens
$\tilde{J}$	Anzahl der Quadraturknoten des groben SDC Propagators
$\hat{J}$	Anzahl der Quadraturknoten des feinen SDC Propagators
$\tilde{U}_{n,j}^k$	Näherung des groben SDC Propagators am Knoten j des Zeitschritts n in der Iteration k
$\hat{U}_{n,j}^k$	Näherung des feinen SDC Propagators am Knoten j des Zeitschritts n in der Iteration k
τ_G	Laufzeit einer einzelnen Ausführung des groben Propagators
τ_F	Laufzeit einer einzelnen Ausführung des feinen Propagators

1 Einleitung

Die vorliegende Arbeit beschäftigt sich mit dem zeitparallelen Algorithmus *Parareal* und dessen Weiterentwicklung, dem *hybriden Parareal/SDC-Algorithmus*. Ziel der Arbeit ist es, den klassischen Parareal-Algorithmus und zwei Varianten des hybriden Parareal/SDC-Algorithmus anhand eigener Implementierungen hinsichtlich der Performance miteinander zu vergleichen. In diesem einleitenden Kapitel wird der Begriff der Zeitparallelität definiert und motiviert. Darüber hinaus wird die historische Entwicklung von zeitparallelen Algorithmen, die mittlerweile etwa fünfzig Jahre umfasst [13], kurz umrissen bevor der Parareal-Algorithmus näher in Kapitel 2 behandelt wird.

1.1 Motivation

Zeitabhängige physikalische Prozesse werden mathematisch i.d.R. durch gewöhnliche und partielle Differentialgleichungen modelliert. Die Berechnung des Zustandes eines solchen zeitabhängigen physikalischen Systems zu einem bestimmten Zeitpunkt erfordert immer die Kenntnis des Zustandes des Systems an einem früheren Zeitpunkt. Dieser Zustand wird als Anfangswert bezeichnet und deswegen nennt man solche Probleme auch Anfangswertprobleme. Dass immer ein früherer Zustand bekannt sein muss, ist durch die allumfassende Kausalität unserer physikalischen Welt eine natürliche Notwendigkeit. Es ist also unmöglich mehr als die unmittelbare Zukunft berechnen zu können. Klassische numerische Verfahren zur approximativen Lösung von Anfangswertproblemen laufen daher seriell in der Zeit ab. Die Zeit wird in einzelne Zeitschritte unterteilt und es wird ein Zeitschritt nach dem nächsten abgearbeitet. Während es zahlreiche Algorithmen für die Parallelisierung von zeitabhängigen partiellen Differentialgleichungen in den Raumdimensionen gibt, sind zeitparallele Algorithmen, also Methoden, die die Lösung von Anfangswertproblemen in mehreren Zeitschritten parallel ermöglichen, im Vergleich dazu wenig verbreitet.

Die Entwicklung der Rechnerarchitekturen in Supercomputern zu immer mehr Rechenkernen hat sich in den letzten Jahren noch beschleunigt. In Abbildung 1.1(a) werden die Anzahl der Kerne des jeweiligen Nummer 1 Supercomputers ab 1993 dargestellt [2]. Es ist ein annähernd exponentieller Anstieg zu erkennen. Diese Entwicklung ist in den letzten Jahren durch das Erreichen der maximal möglichen Taktfrequenzen in einem Prozessor notwendig geworden, da ein Anstieg der Rechenleistung so nur noch durch die Erhöhung

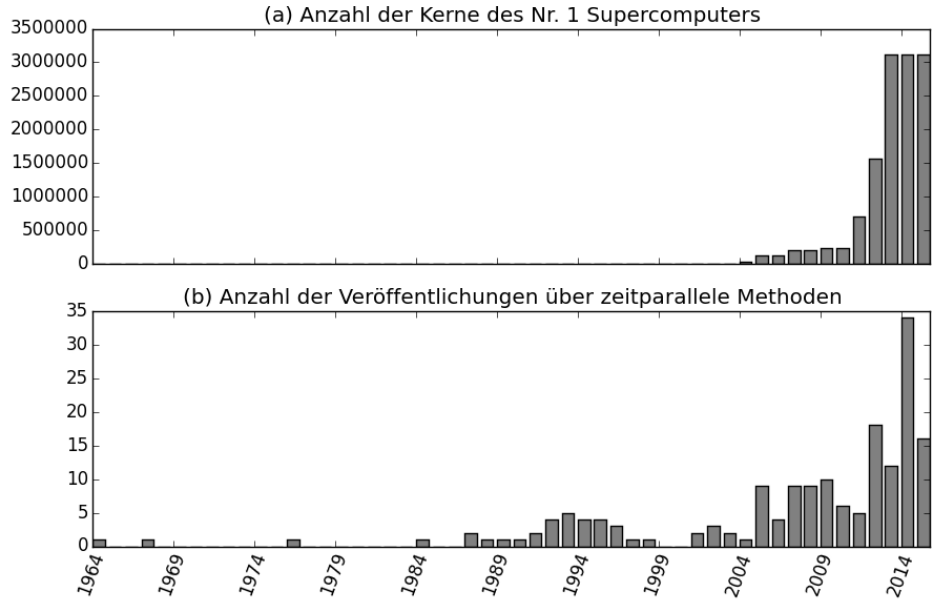

Abbildung 1.1: (a) Anzahl der Kerne des Nr. 1 Supercomputers von 1993 bis 2015 [2]. (b) Anzahl der Veröffentlichungen zum Thema der zeitparallelen Algorithmen [1].

der Anzahl der Kerne ermöglicht werden kann. Die Begrenzung der Taktfrequenzen je Prozessor verbunden mit dem Anstieg der Anzahl der Kerne ist außerdem energetisch effizienter und wird daher in neueren Architekturen angewandt. Diese Entwicklung zu immer mehr Kernen wird sich auch in Zukunft fortsetzten, da nur so die Rechenleistungen erhöht werden können. Um diese immense Anzahl der Rechenkerne, die aktuell bei den schnellsten Supercomputern eine Größenordnung von mehreren Millionen erreicht hat, effizient nutzen zu können, sind Anpassungen bzw. Neuentwicklungen der Software und den numerischen Verfahren von Nöten.

Die räumliche Parallelisierung von zeitabhängigen partiellen Differentialgleichungen stößt durch die begrenzte starke Skalierbarkeit an eine Grenze, ab der eine Zunahme der Kerne für die Berechnung keine Beschleunigung zur Folge hat. Als eine konsequente Antwort auf diese Problemstellung wurden zeitparallele Algorithmen entwickelt, die sich durch die zusätzliche Parallelisierung in der Zeitdimension auszeichnen. Abbildung 1.1(b) illustriert das gestiegene Interesse in zeitparallelen Algorithmen in den letzten Jahren durch die Anzahl der Veröffentlichungen zu dem Thema. Es ist gut zu erkennen, dass die hohe Anzahl der Veröffentlichungen mit dem Anstieg der Kerne in den letzten Jahren korreliert. Das Interesse in zeitparallelen Algorithmen Ende der 80er und Anfang der 90er Jahren flaute wieder ab, da die Ressourcen für die räumliche Parallelisierung verwendet wurden und keine Parallelisierung in der Zeitdimension erforderlich war.

1.2 Entwicklung der zeitparallelen Algorithmen

In diesem Abschnitt wird eine grobe Übersicht über die historische Entwicklung von den zeitparallelen Algorithmen, die im Detail in der Übersichtsarbeit von Gander [13] zusammen gefasst sind, gegeben. Die erste Idee zur Parallelisierung von Anfangswertproblemen in der Zeitdimension wurde schon im Jahre 1984 von Nievergelt [27] hervorgebracht, wobei damals noch kein Bedarf an Zeitparallelisierung bestand obgleich die oben erwähnte Entwicklung vom Autor vorhergesehen wurde. Die in dieser Veröffentlichung beschriebene Methode wurde daher vom Autor als experimentell und als Machbarkeitsstudie eingestuft. Der Algorithmus kann als „mehrfaches Schießverfahren (engl. Multiple Shooting Method)“ [13] kategorisiert werden, wobei darin ausgehend von einer groben Anfangsnäherung parallel auf den einzelnen Zeitschritten genauere Näherungen berechnet werden und diese Näherungen schließlich seriell kombiniert werden.

Dieser Ansatz von Nievergelt wurde von Bellen und Zennaro 1989 [6] aufgegriffen und weiterentwickelt. Die Methode basiert auf der iterativen Lösung einer Fixpunktgleichung über das Steffensen-Verfahren. Dieses Verfahren wird direkt auf dem diskretisierten Anfangswertproblem definiert. Eine Weiterentwicklungen der Ansätze von Nievergelt und Bellen und Zennaro wurden von Chartier et. al. [9] und Saha et. al. 1993 bzw. 1996 veröffentlicht. Beide setzen auf der Fixpunktgleichung auf und definieren eine iterative Lösungsmethode, wobei im Gegensatz zu Bellen und Zennaro die Verfahren auf dem kontinuierlichem Level definiert wird. Saha et. al. wenden die Methode anders als Chartier et. al. auf Hammilton'sche Systeme an. Der 2001 von Lions et. al. entwickelte Parareal-Algorithmus kann als eine spezielle Variante von dem Verfahren von Saha et. al. angesehen werden, wobei hier eine andere Diskretisierung für die Näherung der Jacobi-Matrix verwendet wird. Der Parareal-Algorithmus wird in Kapitel 2 im Detail behandelt.

Gander beschreibt in seiner Arbeit weitere Klassen von zeitparallelen Algorithmen. Zum einen wird die Klasse der „Gebietszerlegungsmethoden in Raum und Zeit“ beschrieben. Eine spezielle Variante dieser Methoden ist die „Waveform Relaxation“-Methode, die ursprünglich für die Simulation von integrierten Schaltkreisen entwickelt wurde. Die Idee hinter dieser Methode ist es, das Problem in einzelne Teilprobleme zu zerlegen. Diese Teilprobleme können dann parallel gelöst werden und die Näherungen werden dann iterativ miteinander gekoppelt, um das ganzheitliche Problem zu lösen. Das zunächst für gewöhnliche Anfangswertprobleme definierte Verfahren wurde in den 90er Jahren auch auf partielle Differentialgleichungen angewandt.

Eine weitere Klasse, die in der Arbeit beschrieben ist, ist die Klasse der zeitparallelen Mehrgitter-Methoden in Raum und Zeit. Die erste Entwicklung dieser Art wurde im Jahre 1984 von Hackbusch [17] veröffentlicht. Klassische Mehrgitterverfahren werden für die iterative Lösung von elliptischen Problemen angewandt, die in jedem einzelnen diskreten Zeitschritt nach der Diskretisierung von parabolischen Problemen entstehen. Im Gegensatz dazu werden die Mehrgitterverfahren in Raum und Zeit direkt auf die

parabolischen Probleme angewandt. Diese Klasse der zeitparallelen Algorithmen wurde intensiv in den 90er Jahren erforscht, wobei dann das Interesse abflachte. Erst in den letzten Jahren wurden neue Algorithmen in dieser Verfahrensklasse, unter anderem der *PFASST*-Algorithmus [11], entwickelt.

Die Arbeit ist folgendermaßen gegliedert: In Kapitel 2 werden der Parareal-Algorithmus und das *Spectral-Deferred-Corrections (SDC)*-Verfahren behandelt, da diese die Grundbausteine des hybriden Parareal/SDC-Verfahrens bilden. Anschließend werden in Kapitel 3 zwei Varianten des hybriden Parareal/SDC-Algorithmus hergeleitet. Außerdem wird die Implementierung des klassischen Parareal und der beiden hybriden Parareal/SDC-Varianten im *PFASST++*-Framework besprochen. In Kapitel 4 werden die Implementierungen anhand des Advektions-Diffusions-Problems auf Konvergenz getestet und die erreichten Speedups werden mit den theoretisch erreichbaren Speedups verglichen. Abschließend werden die Ergebnisse in Kapitel 5 resümiert und die Brücke zum *PFASST*-Algorithmus, die eine konsequente Weiterentwicklung des hybriden Parareal/SDC ist, geschlagen.

2 Parareal und SDC

Die in diesem Kapitel vorgestellten numerischen Verfahren dienen zur Lösung von Anfangswertproblemen, die wie folgt definiert sind:

$$\begin{aligned} u'(t) &= f\left(t, u(t)\right), \qquad t \in [T_0, T], \\ u(T_0) &= u_0, \end{aligned} \tag{2.1}$$

Mit $u_0, u(t) \in \mathbb{C}^N$ und $f : \mathbb{R} \times \mathbb{C}^N \to \mathbb{C}^N$. Das Anliegen dieses Kapitels ist die theoretische Grundsteinlegung für den hybriden Parareal/SDC-Algorithmus, der in Kapitel 3 besprochen wird. Es wird zunächst in Unterkapitel 2.1 der zeitparallele Algorithmus *Parareal* vorgestellt und anschließend wird in Unterkapitel 2.2 das Fehlerkorrekturverfahren SDC (*Spectral Deferred Correction*) besprochen, da diese beiden Verfahren bei dem hybridem Parareal/SDC-Algorithmus zur Steigerung der parallelen Effizienz kombiniert werden

2.1 Parareal

In diesem Unterkapitel wird zunächst die Entwicklungsgeschichte von Parareal rekapituliert und dabei einige Anwendungsfälle, auf die Parareal in der aktuellen Forschung erfolgreich angewandt wurde, angesprochen. Anschließend werden die Grundlagen sowie die mathematische Formulierung von Parareal besprochen um die theoretisch erreichbare parallele Effizienz herzuleiten. Die Herleitung wird im Kapitel 3 aufgegriffen, wenn die parallele Effizienz des hybriden Parareal/SDC-Algorithmus thematisiert wird, um die dadurch erreichbare parallele Effizienzsteigerung zu verdeutlichen.

2.1.1 Entwicklung und Anwendungen

Der zeitparallele Algorithmus *Parareal* wurde im Jahre 2001 von Lions et al. [22] entwickelt, mit dem Ziel, Anfangswertprobleme durch die zusätzliche Parallelisierung in Zeitrichtung in Echtzeit berechnen zu können. Aus dieser Motivation entstand die Namensgebung für den Algorithmus, *Parareal*, eine Zusammensetzung aus *Parallel* und *Realtime*. Eine Konvergenzanalyse von Parareal wurde 2008 in [14] veröffentlicht, in

dem die superlineare Konvergenz des Algorithmus für ein System von nicht linearen gewöhnlichen Differentialgleichungen mathematisch nachgewiesen und mithilfe verschiedener Beispielprobleme, wie die Lorenz-Gleichung, numerisch gezeigt wurde. Die Stabilität von Parareal für autonome Systeme wurde in [36] untersucht. Für Systeme mit reellen Eigenwerten ist die Stabilitätsbedingung erfüllt. Für Systeme mit imaginären Eigenwerten wurde numerisch die Instabilität von Parareal nachgewiesen.

Ein wichtiges Einsatzgebiet für den Parareal-Algorithmus sind Wetterberechnungen, für die der Zeitgewinn durch die zusätzliche Parallelisierung in Zeitrichtung eine kritische Bedeutung hat. Parareal wurde auch auf viele andere Probleme erfolgreich angewandt und getestet [3, 5, 20, 21, 23, 29]. Im folgenden werden einige aktuelle Anwendungen von Parareal angesprochen, um die anhaltende Relevanz des Algorithmus zu verdeutlichen.

Ariel. et. al. [3] setzten Parareal erfolgreich für die Lösung von stark oszillierenden gewöhnliche Differentialgleichungen ein, in dem sie den Parareal-Algorithmus mit Lösern für Mehrskalen-Berechnungen kombinierten. Parareal wurde von Kreienbuehl et. al. [21] für die zeitparallele Simulation der Diffusion durch die Haut angewandt. In der Veröffentlichung von Ruprecht et. al. [29] wurde der Algorithmus auf Diffusionsprobleme, wie die Wärmeleitungsgleichung angewandt und gute Konvergenzeigenschaften durch das Varieren der Problem-Koeffizienten konnten nachgewiesen werden. In [20] wurde Parareal mit einem raumparallelen, zeitlich seriellen Löser für die Lösung der Burger's-Gleichung kombiniert und es wurde eine Steigerung des Speedups jenseits der Sättigungsgrenze des Speedups des raumparallelen Lösers gemessen. Baudron et. al. [5] setzten Parareal erfolgreich für die Zeitparallelisierung eines Lösers für die Neutronen-Diffusionsgleichung ein. Loderer et. al. [23] wandten Parareal für die Echtzeitsimulation in der Automobilbranche ein.

Diese zahlreichen Anwendungen von Parareal illustrieren dessen vielseitige Einsetzbarkeit. Die Flexibilität des Algorithmus entspringt aus dessen mathematischer Formulierung, die den Anwendern viele Freiheiten lässt, den Algorithmus für verschiedene Problemstellungen anzupassen bzw. einzusetzen. Diese mathematische Formulierung wird im folgenden Abschnitt behandelt.

2.1.2 Grundlagen

Die Zeit hat im Gegensatz zu anderen physikalischen Größen eine unumkehrbare Richtung. Daher werden zeitabhängige dynamische Prozesse, die durch Differentialgleichungen mathematisch modelliert werden, intuitiv als sequentiell betrachtet. Die Konstruktion von parallelen Algorithmen für die Lösung von Anfangswertproblemen erscheint also zunächst als ein schwieriges Unterfangen, da die Lösung an einem Zeitpunkt stets von den Lösungen an früheren Zeitpunkten abhängt. Ein klassisches Gebiet, in dem die Parallelisierung erfolgreich angewandt wird, ist die Lösung von Randwertproblemen. Die

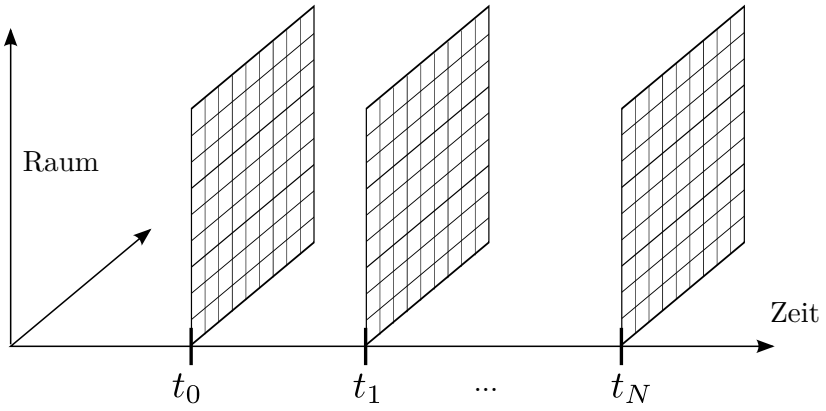

Abbildung 2.1: Graphische Darstellung der Zeiteinteilung.

hier angewandte Methode heißt *Gebietszerlegung* (engl. *Domain Decomposition*) [31], die die Parallelisierung durch die Zerlegung des Problemgebietes in einzelne Teilgebiete, auf denen die Lösungen unabhängig voneinander berechnet werden können, ermöglicht. Durch ein iteratives Verfahren werden die Randwerte an den Gebietsgrenzen zwischen angrenzenden Teilgebieten ausgetauscht. Da die Methode der Gebietszerlegung die Parallelisierung über die räumliche Aufteilung des zu lösenden Problems realisiert, spricht man von *räumlicher Parallelisierung*. Für zeitabhängige Probleme wie Anfangswertprobleme gibt es eine weitere Möglichkeit für die Parallelisierung, nämlich analog zu der Methode der Gebietszerlegung, die Aufteilung des zu lösenden Zeitbereichs, um unabhängig voneinander lösbare Teilprobleme zu generieren.

Beim Parareal-Algorithmus wird das Prinzip der Gebietszerlegung auf die Zeit angewandt, indem die zu berechnende Zeitspanne $[T_0, T]$ in N Intervalle $[t_n, t_{n+1}]$, mit den Intervallgrenzen

$$T_0 = t_0 < t_1 < \cdots < t_N = T$$

unterteilt wird. Diese Unterteilung der Zeit-Achse wird graphisch in Abbildung 2.1 dargestellt. Dabei wird die räumliche Diskretisierung als ein Rechteckgitter angedeutet. Auf jedem dieser Intervalle kann jeweils unabhängig ein Anfangswertproblem gelöst werden. Um diese Anfangswertprobleme lösen zu können, werden an den Intervallanfängen Anfangswerte benötigt. Diese Anfangswerte werden über die serielle Ausführung eines schnellen, dafür aber ungenauen Lösungsverfahrens berechnet. Anschließend können die einzelnen Anfangswertprobleme unabhängig voneinander mit berechnungsintensiven aber genaueren Lösungsverfahren gelöst werden. Parareal geht iterativ vor und korrigiert die seriell berechneten ungenauen Näherungen durch die parallel berechneten genaueren Näherungen über eine Iterationsvorschrift. Der Algorithmus besteht also im Wesentlichen aus zwei Komponenten, nämlich der seriellen Ausführung eines schnellen, aber ungenauen Lösers und der parallelen Ausführung eines genauen, dafür aber langsameren Lösers. Diese zwei Komponenten sind im Grunde genommen Einschrittverfahren, über die eine Näherung der Lösung $U_n \sim u(t_n)$ zu einem bestimmten Zeitpunkt t_n bei

Angabe eines Anfangswertes U_0 berechnet werden können. Diese beiden Einschrittverfahren werden im Folgenden als *Propagatoren* bezeichnet, angelehnt an das englische „to propagate“ (ausbreiten), da die numerische Lösung in positiver Zeitrichtung „ausgebreitet“ wird. Die Charakterisierung dieser beiden Propagatoren geschieht wie angedeutet über den numerischen Berechnungsaufwand. Der seriell ausgeführte Propagator ist ein schnelles Einschrittverfahren, wie z.B. das Euler-Verfahren:

$$G(t_{n+1}, t_n, U_n) = U_n + \Delta t f(U_n, t_n)$$

Da dieser Propagator eine grobe Näherung erzielt, wird dieser als *grober Propagator* $G(t_{n+1}, t_n, U_n)$ bezeichnet. Im Gegensatz dazu ist der zweite Propagator ein Einschrittverfahren, das eine sehr viel genauere Näherung erzielt, dabei aber höhere Berechnungskosten aufweist. Dieser Propagator wird als *feiner Propagator* $F(t_{n+1}, t_n, U_n)$ bezeichnet. Die höhere Genauigkeit des feinen Propagators wird beispielsweise durch die Verwendung einer kleineren Schrittweite als beim groben Propagator ermöglicht. Es ist auch möglich ein anderes Lösungsverfahren mit einer höheren Fehlerordnung als das Verfahren des groben Propagators zu verwenden. Der Parareal-Algorithmus lässt hier verschiedene Möglichkeiten für die Definition der Propagatoren zu und ist dadurch sehr flexibel einsetzbar. Es ist nur wichtig, dass der serielle Anteil des Algorithmus so gering wie möglich ausfällt, damit eine gute parallele Effizienz erreicht wird. Auf die Bedeutung der Wahl der Propagatoren für die parallele Effizienz wird in Kapitel 2.1.4 genauer eingegangen.

Im Folgenden wird der Parareal-Algorithmus zum besseren Verständnis der formellen Definition verbal beschrieben. Es wird jedem Prozess jeweils ein Zeitintervall, wie oben definiert, zugewiesen. Dabei werden die Prozesse folgendermaßen als $P_0 \dots P_{N_p-1}$ durchnummeriert. In der Initialisierungsphase erhalten alle Prozesse die Anfangswerte U^0_{n+1} durch die serielle Ausführung des groben Propagators:

$$U^0_{n+1} = G(t_{n+1}, t_n, U^0_n), \qquad n = 0, \dots, N_p - 1 \tag{2.2}$$

Anschließend kann jeder Prozess unabhängig voneinander mit Hilfe des feinen Propagators und des Anfangswerts eine genauere Näherung für den nächsten Zeitpunkt berechnen. Prozess 0 hat somit die beste in dem ersten Zeitintervall erzielbare Näherung berechnet und sendet die Näherung als U^1_1 an Prozess 1 weiter. Prozess 1 empfängt diese Näherung als neuen Anfangswert. Mit dem neuen Anfangswert wird anschließend über den groben Propagator eine Näherung zum nächsten Zeitpunkt berechnet. Nun hat Prozess 1 drei Näherungen für den nächsten Zeitpunkt, nämlich die Näherung des groben Propagators U^0_2, die in der Initialisierungsphase berechnet wurde, die Näherung des feinen Propagators, die mit dem letzten (ungenaueren) Anfangswert U^0_1 berechnet wurde und die Näherung des groben Propagators mit dem neuen (genaueren) U^1_1 Anfangswert. In Abbildung 2.2 werden die Parareal-Iterationen zum besseren Verständnis in einem Dreiecksschema dargestellt. Dabei wird die Ausführung des feinen Propagators als roter Pfeil und die Ausführung des groben Propagators als grüner Pfeil verbildlicht. Der Ursprung eines Pfeils ist jeweils am Anfangswert mit dem der jeweilige Propagator

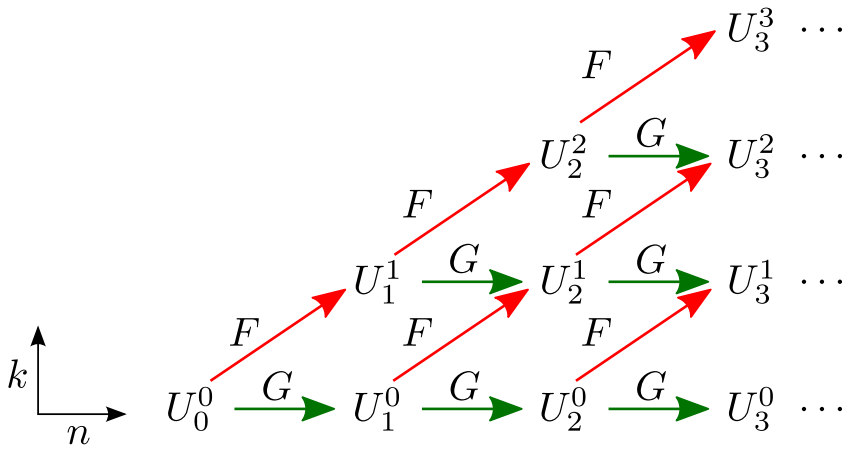

Abbildung 2.2: Schematische Darstellung der Parareal-Iterationen in einem Dreiecksschema.

ausgeführt wird.

Es stellt sich nun die Frage, welche Näherung an den nächsten Prozess gesendet werden soll. Die Näherung des feinen Propagators hat den Makel, dass sie mit dem alten ungenauen Anfangswert berechnet wurde. Die aktuelle Näherung des groben Propagators wurde zwar mit dem neuen genaueren Anfangswert berechnet, aber der grobe Propagator erzeugt eine schlechtere Näherung als der feine Propagator.

Die zentrale Idee von Parareal ist nun, diese Näherungen zu kombinieren und so als neuen Anfangswert an den nächsten Prozess zu schicken. Dazu wird in jeder Parareal-Iteration zur Näherung des feinen Propagators die Differenz der aktuellen Näherung $G(t_{n+1}, t_n, U_n^{k+1})$ und der letzten Näherung $G(t_{n+1}, t_n, U_n^k)$ des groben Propagators addiert, wodurch die Verbesserung durch den neuen Anfangswert auf die Näherung des feinen Propagators $F(t_{n+1}, t_n, U_n^k)$ angewandt wird. Die somit erzielte Näherung U_{n+1}^{k+1} wird an den nächsten Prozess weitergesendet, wodurch seriell in Zeitrichtung die Verbesserung ausgebreitet („propagiert“) wird. Diese Iterationsvorschrift wird formell folgendermaßen definiert:

$$\begin{aligned} U_{n+1}^{k+1} &= F(t_{n+1}, t_n, U_n^k) + G(t_{n+1}, t_n, U_n^{k+1}) - G(t_{n+1}, t_n, U_n^k), \\ U_0^0 &= u_0, \qquad n = 0, \ldots, N_p - 1, \qquad k = 0, \ldots, K - 1. \end{aligned} \tag{2.3}$$

Dabei wird für $k = 0$ die Initialisierung von Parareal mit dem groben Propagator wie in Gleichung (2.2) ausgeführt. Parareal führt diese Korrektur nun Iterativ fort. In jeder Iteration führen die Prozesse jeweils parallel den feinen Propagator aus und seriell den groben, um die Korrektur zu berechnen.

Da Parareal ein iteratives Verfahren ist, braucht es eine Abbruchbedingung. Nach k Iterationen von Parareal ist die Näherung U_n^k für $n \leq k$ genau gleich der Näherung, die durch

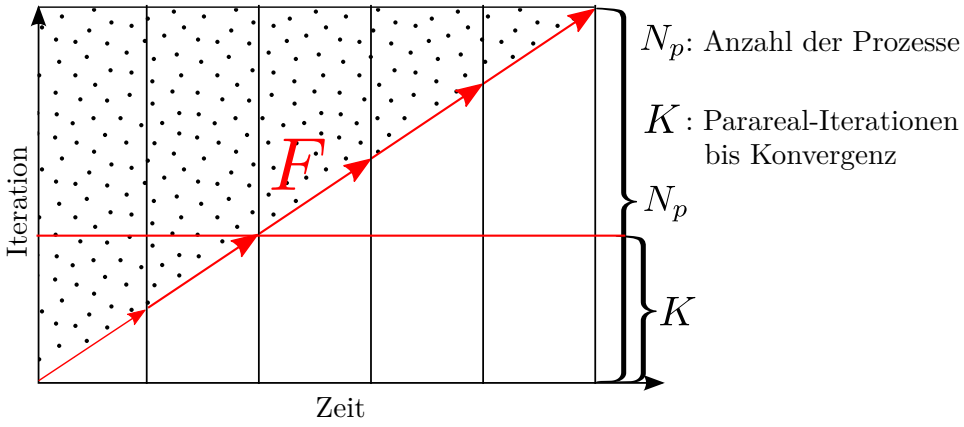

Abbildung 2.3: Graphische Veranschaulichung der Bedingung für die parallele Effizienz von Parareal. Die Ausführung des feinen Propagators wird durch die roten Pfeile dargestellt. Die horizontale Linie beschreibt das des Konvergenzkriteriums.

die sequentielle Ausführung des feinen Propagators zu erreichen gewesen wäre. Daher ist nach N_p Iterationen die Näherung am Endzeitpunkt $U_{N_p}^{N_p}$ identisch mit der durch die reine serielle Ausführung des feinen Propagators berechnete Näherung. Aufgrund dessen ist nur dann eine parallele Effizienz von Parareal gegenüber der seriellen Ausführung des feinen Propagators gegeben, wenn die Anzahl von maximalen Iterationen K, bis ein vorgegebenes Konvergenzkriterium erreicht ist, deutlich kleiner ist als N_p, und wenn der numerische Aufwand des groben Propagators deutlich geringer ist als der Aufwand des feinen Propagators. Ein einfaches Konvergenzkriterium für die Verwendung einer Abbruchbedingung wäre zum Beispiel, dass die Differenz der Näherung der aktuellen Iteration und die Näherung der letzten Iteration $U_n^k - U_n^{k-1}$ kleiner ist als ein vorgegebenes ϵ. Die Bedeutung einer Abbruchbedingung wird in Abbildung 2.3 veranschaulicht. Darin werden die Parareal-Iterationen wie in Abbildung 2.2 als Dreiecksschema dargestellt, wobei der feine Propagator gesondert mit roten Pfeilen veranschaulicht wird. Parareal erreicht nach N_p Iterationen die gleiche Näherung wie der seriell ausgeführte feine Propagator. Dadurch, dass Parareal zusätzlich zum feinen Propagator je Iteration auch noch den groben Propagator ausführt, ist bei N_p Iterationen keine parallele Effizienz zu erwarten. Dies wird auch durch das Dreiecksschema deutlich, da die Hälfte der N_p Prozesse bei N_p-Iterationen im Leerlauf sind. Damit eine möglichst hohe parallele Effizienz erreicht wird, muss die maximale Anzahl der benötigten Iterationen K für die Konvergenz möglichst niedrig sein. Die formelle Herleitung der parallelen Effizienz wird in Abschnitt 2.1.4 thematisiert. Davor wird im nächsten Abschnitt eine Umformulierung von Parareal definiert, die eine räumliche Vergröberung für den groben Propagator ermöglicht. Auf die Vergröberungsstrategien wird in Abschnitt 3.1.5 in Kapitel 3 näher eingegangen.

2.1.3 Räumliche Vergröberung

Für zeitabhängige partielle Differentialgleichungen bietet Parareal die Möglichkeit, eine Vergröberung im Raum durchzuführen. Dies geschieht durch die Verwendung von verschiedenen räumlichen Diskretisierungen für die beiden Propagatoren, wobei für den groben Propagator weniger Freiheitsgrade verwendet werden als für den feinen Propagator. Bei gitterbasierter Diskretisierung kann beispielsweise für den groben Propagator ein Gitter mit einer geringeren Auflösung verwendet werden. Parareal benutzt für die Näherungen U_n^k die räumliche Diskretisierung des feinen Propagators. Da die Ergebnisse des groben Propagators eine geringere räumliche Auflösung haben, entsteht für die Berechnung der Iterationsvorschrift (2.3) die Notwendigkeit von Transferoperationen zwischen den räumlichen Diskretisierungen. Für den Transfer vom groben zum feinen Gitter wird ein *Interpolationsoperator* $\mathbf{P}$ benötigt. Für die umgekehrte Richtung, also dem Transfer vom feinen zum groben Gitter wird ein *Restriktionsoperator* $\mathbf{R}$ verwendet. Die Konvergenz von Parareal kombiniert mit räumlicher Vergröberung wurde von Ruprecht in [28] untersucht. In dieser Veröffentlichung wurde besonders der Einfluss der Interpolationsordnung auf die Konvergenz hervorgehoben. Die Einführung der Transferoperatoren für die räumliche Vergröberung macht eine geringfügige Umformulierung der Iterationsvorschrift von Parareal (2.3) notwendig:

$$U_{n+1}^{k+1} = F(t_{n+1}, t_n, U_n^k) + \mathbf{P}G(t_{n+1}, t_n, \mathbf{R}U_n^{k+1}) - \mathbf{P}G(t_{n+1}, t_n, \mathbf{R}U_n^k) \tag{2.4}$$

Durch diese Umformulierung ergibt sich die Möglichkeit der räumlichen Vergröberung, wodurch sich der Berechnungsaufwand des groben Propagators noch mehr verringern lassen kann. Der serielle Anteil von Parareal kann dadurch zusätzlich verkleinert werden, wodurch die parallele Effizienz sich steigern kann.

2.1.4 Parallele Effizienz

In diesem Abschnitt geht es um die theoretische parallele Effizienz von Parareal und deren Herleitung. Die folgenden Ausführungen basieren auf der Annahme, dass die Prozessoren identisch sind und dass die Kommunikation vernachlässigt werden kann. Außerdem wird die Ausführungszeit für die Berechnung der Summe in der Iterationsvorschrift (2.3) vernachlässigt. Die Zeit für die Ausführung des groben Propagators wird als τ_G und die Zeit für die Ausführung des feinen Propagators wird als τ_F bezeichnet. Die Anzahl der Prozesse wird mit N_p bezeichnet und die zu berechnende Zeitspanne $[0, T]$ wird in gleich große Zeitintervalle mit der Schrittweite $\Delta T = T/N_p$ unterteilt. Bei der Initialisierungsphase wird der grobe Propagator seriell ausgeführt, sodass sich eine Ausführungszeit von $N_p\tau_G$ ergibt. In jeder weiteren Iteration führt jeder Prozess jeweils einmal den feinen und groben Propagator aus, sodass die Ausführungszeit pro Parareal-Iteration $\tau_G + \tau_F$ beträgt. Die gesamte Ausführungszeit ist abhängig von der Anzahl der benötigten Iterationen K und beträgt somit $N_p\tau_G + K(\tau_G + \tau_F)$. Die Ausführungszeiten

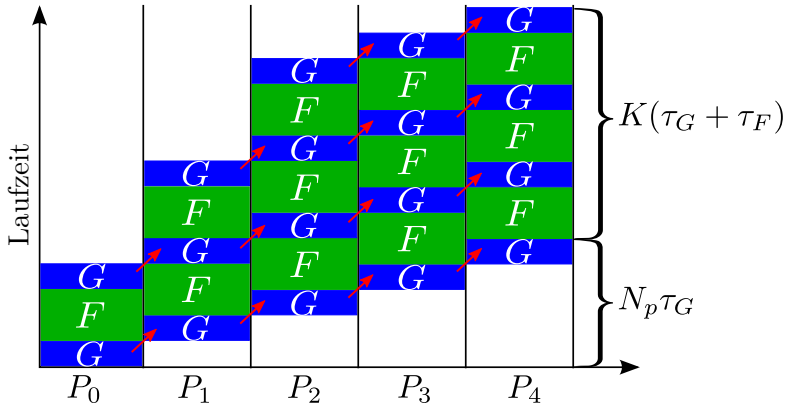

Abbildung 2.4: Graphische Darstellung der Laufzeit von Parareal

werden in Abbildung 2.4 graphisch veranschaulicht.

Der theoretische Speedup wird gegenüber der seriellen Ausführungszeit des feinen Propagators gemessen, sodass sich folgender Speedup ergibt:

$$S = \frac{N_p\tau_F}{N_p\tau_G + K(\tau_G + \tau_F)} = \frac{1}{\frac{\tau_G}{\tau_F} + \frac{K}{N_p}(\frac{\tau_G}{\tau_F} + 1)}. \tag{2.5}$$

Durch die Einführung eines Faktors $\alpha = \tau_G/\tau_F$, der das Verhältnis der Ausführungszeiten des groben bzw. des feinen Propagators ausdrückt, vereinfacht sich die Formel zu

$$S = \frac{1}{\alpha + \frac{K}{N_p}(\alpha + 1)}. \tag{2.6}$$

Die daraus resultierende parallele Effizienz $E = S/N_p$ ist dann

$$E = \frac{1}{\alpha \cdot N_p + K(\alpha + 1)}. \tag{2.7}$$

Die parallele Effizienz von Parareal hängt demnach von zwei Einflussfaktoren ab, nämlich dem Vergröberungsfaktor α, also das Laufzeitverhältnis zwischen dem feinen und groben Propagator, und die benötigte Anzahl an Iterationen K. Diese beiden Faktoren haben auch eine Abhängigkeit untereinander, da der Vergröberungsfaktor α einen wesentlichen Einfluss auf die Iterationszahl K hat. Je kleiner α ist, also je gröber und somit ungenauer der grobe Propagator im Vergleich zum feinen Propagator ist, desto mehr Iterationen werden benötigt, um die gewünschte Konvergenzgenauigkeit zu erreichen. Dies geht aus den numerischen Ergebnissen hervor, die in Kapitel 4 behandelt werden. Aus der Gleichung für die Effizienz (2.7) lässt sich auch eine obere Schranke ablesen, denn es gilt $E < 1/K$. Auch wenn α durch eine starke Vergröberung sehr klein wird, kann die parallele Effizienz nicht über diese Schranke verbessert werden. Da mindestens zwei Parareal-Iterationen ausgeführt werden müssen, um ein Konvergenzkriterium anwenden

zu können, ist die parallele Effizienz von Parareal immer kleiner als $\frac{1}{2}$. Aus diesem Umstand entstand auch die Motivation, den hier vorgestellten Algorithmus hinsichtlich der Steigerung der parallelen Effizienz weiterzuentwickeln. Doch bevor die Weiterentwicklung von Parareal in Form des hybriden Parareal/SDC in Kapitel 3 behandelt wird, wird im nächsten Unterkapitel das iterative Verfahren der sogenannten *Spectral Deferred Correction* für die Lösung von Anfangswertproblemen, das sich gut für die Wahl als Propagator eignet, thematisiert.

2.2 Spectral Deferred Correction

In diesem Kapitel wird das Spectral Deferred Correction Verfahren (SDC) vorgestellt. Zunächst werden die Grundlagen des Verfahrens in Abschnitt 2.2.1 hergeleitet und eine einfache Diskretisierung entwickelt. Anschließend wird in Abschnitt 2.2.2 eine semi-implizite Variante vorgestellt, die sich auch für die Lösung von steifen Anfangswertproblemen eignet und deswegen eine höhere Relevanz in der Praxis hat. In Abschnitt 2.2.3 werden die wichtigsten Eigenschaften wie die Stabilität sowie der numerische Aufwand von SDC zusammengefasst. Abschließend werden in 2.2.4 aktuelle Weiterentwicklungen von SDC rekapituliert.

2.2.1 Grundlagen

Das Spectral Deferred Correction Verfahren ist eine Abwandlung der in den 1970er Jahren entwickelten Deferred Correction Methoden [7], die im Jahre 2000 von Dutt et. Al [10] veröffentlicht wurde. Es handelt sich dabei um ein iteratives Lösungsverfahren für gewöhnliche Anfangswertprobleme (2.1), das vorteilhafte Eigenschaften wie eine gute Stabilität und eine nahezu beliebig hohe Ordnung besitzt. Die Konstruktion des Verfahrens basiert auf der Picard-Integralformulierung des Anfangswertproblems (2.1), die folgendermaßen definiert ist:

$$\begin{aligned} u(t) &= u_0 + \int_{T_0}^{t} f(\tau, u(\tau)) d\tau, \\ u_0 &= u(T_0). \end{aligned} \tag{2.8}$$

Für die numerische Berechenbarkeit der Gleichung wird ein einzelner Zeitschritt $[t_n, t_{n+1}]$ der Größe $\Delta t = t_{n+1} - t_n$ in mehrere Zwischenschritte unterteilt. Dies geschieht durch die Einführung einer Anzahl von in diesem Zeitschritt liegenden Zeitpunkten, genannt Knoten, $\mathbf{t}_n = [t_1, \ldots, t_J]$ mit $t_n \leq t_1 < \cdots < t_J \leq t_{n+1}$, wobei sich hierbei die Knoten $\mathbf{t}_n$ im Gegensatz zu klassischen Deferred Correction Verfahren auf die Knoten der Gauß-Quadratur beziehen. Der Zusatz *Spectral* im Namen des Verfahrens bezieht sich genau auf diese Eigenschaft, da im englischen die Gauß-Quadratur auch *spectral integration*

genannt wird [15]. Die Knoten bzw. die Zwischenzeitschritte werden mit dem Index j mit $j \in [1, \ldots, J]$ indiziert. Die numerischen Näherungen an den Zwischenzeitschritten werden ebenfalls mit j indiziert. Angefangen mit dem Anfangswert $U_{n,1} \approx u(t_n)$ zum Zeitschritt t_n, wird eine initiale Näherung $\mathbf{U}_n^0 = [U_{n,1}^0, \ldots, U_{n,J}^0]$ an den Zwischenzeitschritten mit einem Standardverfahren wie z.B. dem Euler-Verfahren berechnet. Über diese Anfangsnäherung kann dann die Fehlergleichung definiert werden:

$$\delta(t) = u(t) - U_n^0(t). \tag{2.9}$$

Der Term $U_n^0(t)$ ist dabei die durch Interpolation ermittelte kontinuierliche Entsprechung zu den diskreten Näherungen $\mathbf{U}_n^0$. Für die Fehlergleichung wird ähnlich wie in (2.8) eine Integralformulierung hergeleitet. Für $u(t)$ in (2.9) wird die Integralformulierung eingesetzt:

$$\delta(t) = u_0 + \int_{T_0}^{t} f\left(\tau, u(\tau)\right) d\tau - U_n^0(t). \tag{2.10}$$

Durch die Addition und Subtraktion des Terms $\int_{T_0}^{t} f\left(\tau, U_n^0(\tau)\right) d\tau$ wird die Gleichung folgendermaßen umformuliert:

$$\delta(t) = \int_{T_0}^{t} \left[f\left(\tau, u(\tau)\right) - f\left(\tau, U_n^0(\tau)\right)\right] d\tau + u_0 + \int_{T_0}^{t} f\left(\tau, U_n^0(\tau)\right) d\tau - U_n^0(t). \tag{2.11}$$

Mit dem Residuum

$$\epsilon(t) = u_0 + \int_{T_0}^{t} f\left(\tau, U_n^0(\tau)\right) d\tau - U_n^0(t) \tag{2.12}$$

wird die Gleichung (2.11) vereinfacht zu:

$$\delta(t) = \int_{T_0}^{t} \left[f\left(\tau, U_n^0(\tau) + \delta(\tau)\right) - f\left(\tau, U_n^0(\tau)\right)\right] d\tau + \epsilon(t) \tag{2.13}$$

Der Term $\epsilon(t)$ kann über die Gauß-Quadratur genau und stabil approximiert werden, da die Anfangsnäherung $U_n^0(t)$ an den Quadraturknoten schon bekannt ist. Die Fehlergleichung (2.13) wird aufgrund einer besseren Handhabbarkeit zu einer Inkrementfunktion umformuliert, die den Fehler an den Knoten $\mathbf{t}_n$ bestimmt:

$$\delta(t_{j+1}) = \delta(t_j) + \int_{t_j}^{t_{j+1}} \left[f\left(\tau, U_n^0(\tau) + \delta(\tau)\right) - f\left(\tau, U_n^0(\tau)\right)\right] d\tau + \epsilon(t_{j+1}) - \epsilon(t_j). \tag{2.14}$$

Die Diskretisierung der Gleichung (2.14) erfordert die Näherung des Integralterms in

$$\epsilon(t_{j+1}) - \epsilon(t_j) = \int_{t_j}^{t_{j+1}} f\left(\tau, U_n^0(\tau)\right) d\tau - U_n^0(t_{j+1}) + U_n^0(t_j). \tag{2.15}$$

Der Integralterm wird über die Gauß-Quadratur an den Knoten $\mathbf{t}_n$ approximiert. Die Approximation kann als Matrix-Vektor Multiplikation mit einer vorberechneten Inte-

grationsmatrix S_j^{j+1} ausgeführt werden:

$$S_j^{j+1} f(\mathbf{t}_n, \mathbf{U}_n^0) \approx \int_{t_j}^{t_{j+1}} f\left(\tau, U_n^0(\tau)\right) d\tau. \tag{2.16}$$

Für die restliche Diskretisierung muss die Fehlergleichung (2.9) noch mittels eines einfachen Einschrittverfahrens an den Knoten $\mathbf{t}_n$ approximiert werden. Beispielsweise wird das explizite Euler-Verfahren angewandt, wodurch sich folgende Inkrementformulierung für die Näherung des Fehlers an den Knoten ergibt:

$$\begin{aligned} \delta_{j+1}^0 = \delta_j^0 + \Delta t_j \left[f\left(t_j, U_{n,j}^0 + \delta_j^0\right) - f(t_j, U_{n,j}^0) \right] \\ + S_j^{j+1} f(\mathbf{t}_n, \mathbf{U}_n^0) - U_{n,j+1}^0 + U_{n,j}^0. \end{aligned} \tag{2.17}$$

SDC ist in dem Sinne flexibel, dass hier anstelle des expliziten Eulers auch Verfahren höherer Ordnung für die Approximation des Fehlers verwendet werden können. Nachdem die numerische Approximation des Fehlers an den Knoten inkrementell ermittelt wurde, kann die Anfangsnäherung der Lösung $\mathbf{U}_n^0$ über $U_{n,j}^1 = U_{n,j}^0 + \delta_{n,j}^0$ korrigiert bzw. verbessert werden. Darin liegt der Ursprung für die Bezeichnung "Deferred Correction" (engl. für verzögerte Korrektur), da die Korrektur der Approximation immer in der nächsten Iteration, also "verzögert" angewandt wird. Die so korrigierte Approximation dient als Anfangsnäherung für die nächste Iteration, in der nochmals der Fehler approximiert wird, um so die nächste Korrektur zu bestimmen. Die Iterationsvorschrift für das SDC-Verfahren lautet dann

$$U_{n,j}^{k+1} = U_{n,j}^k + \delta_{n,j}^k. \tag{2.18}$$

wobei k der Iterationsindex ist. In jeder Iteration wird die Näherung der Lösung gemäß der Konvergenzordnung der verwendeten Näherung für die Fehlergleichung in Gleichung (2.17) verbessert, vorausgesetzt die Quadraturregel in Gleichung (2.16) ist genau genug. Mit der in Gleichung (2.17) verwendeten Approximation erster Ordnung, hat die Näherung der Lösung nach K Iterationen K-te Ordnung. Durch die Umformulierung der Iterationsvorschrift (2.18) können die $U_{n,j}^{k+1}$ auch direkt berechnet werden. In die Korrekturgleichung

$$U_{n,j+1}^{k+1} = U_{n,j+1}^k + \delta_{n,j+1}^k \tag{2.19}$$

wird $\delta_{n,j+1}^k$ durch die diskrete Inkrementformulierung (2.17) ersetzt und man erhält:

$$\begin{aligned} U_{n,j+1}^{k+1} = U_{n,j+1}^k + \delta_{n,j}^k + \Delta t_j \left[f\left(t_j, U_{n,j}^k + \delta_{n,j}^k\right) - f(t_j, U_{n,j}^k) \right] \\ + S_j^{j+1} f(\mathbf{t}_n, \mathbf{U}_n^k) - U_{n,j+1}^k + U_{n,j}^k. \end{aligned} \tag{2.20}$$

Der Term $U_{n,j}^k + \delta_{n,j}^k$ kann durch $U_{n,j}^{k+1}$ ersetzt werden. Außerdem gilt für $\delta_{n,j}^k$ über die Korrekturgleichung (2.18) $\delta_{n,j}^k = U_{n,j}^{k+1} - U_{n,j}^k$. Durch diese Ersetzungen in (2.20) fallen die Terme $U_{n,j+1}^k$ und $U_{n,j}^k$ weg und es bleibt die direkte Inkrementformulierung für das

SDC-Verfahren stehen:

$$U_{n,j+1}^{k+1} = U_{n,j}^{k+1} + \Delta t_j \left[f(t_j, U_{n,j}^{k+1}) - f(t_j, U_{n,j}^{k}) \right] + S_j^{j+1} f(\mathbf{t}_n, \mathbf{U}_n^k). \tag{2.21}$$

2.2.2 Semi-implizite Formulierung

Anstelle des expliziten Eulers kann für die Diskretisierung in Gleichung (2.17) auch das implizite Euler-Verfahren verwendet werden. Das SDC-Verfahren eignet sich besonders gut für die Zeitintegration von Anfangswertproblemen, die in steife und nicht-steife Terme aufgeteilt werden kann. Aufgrund der schlechten Stabilität von expliziten Verfahren bei steifen Anfangswertproblemen werden in der Regel implizite Verfahren eingesetzt. Wenn sowohl steife und nicht-steife Terme in der Gleichung auftauchen ist es deswegen oft viel effizienter, nur die nicht-steifen Terme explizit zu behandeln und die steifen Terme implizit. Solche Verfahren werden semi-implizite oder IMEX (engl. *implicit-explicit*) Verfahren genannt. Die semi-implizite Formulierung von SDC wurde zuerst von Minion 2003 beschrieben [24].

Nun wird die IMEX bzw. semi-implizite Variante von SDC hergeleitet. Dafür wird das Anfangswertproblem etwas umformuliert:

$$\begin{aligned} u'(t) =& f\left(t, u(t)\right) = f_E\left(t, u(t)\right) + f_I\left(t, u(t)\right), \qquad t \in [T_0, T] \\ u(T_0) =& u_0 \end{aligned} \tag{2.22}$$

Dabei steht $f_E(t, u(t))$ für den nicht-steifen und deswegen explizit zu behandelnden Anteil der Gleichung und $f_I(t, u(t))$ dementsprechend für den steifen, implizit zu behandelnden Anteil. Ein semi-implizites Verfahren erster Ordnung für die Berechnung einer initialen Näherung ist wie folgt definiert:

$$U_{n,j+1}^0 = U_{n,j}^0 + \Delta t_j (f_E(t_j, U_{n,j}^0) + f_I(t_{j+1}, U_{n,j+1}^0)) \tag{2.23}$$

Wie bei der Herleitung der expliziten Formulierung von SDC wird auch für das semi-implizite SDC die Fehlergleichung aufgestellt:

$$\begin{aligned} \delta(t) = \int_{T_0}^{t} [& f_E(\tau, U_n^0(\tau) + \delta(\tau)) - f_E(\tau, U_n^0(\tau)) \\ & + f_I(\tau, U_n^0(\tau) + \delta(\tau)) - f_I(\tau, U_n^0(\tau))] d\tau + \epsilon(t), \end{aligned} \tag{2.24}$$

wobei die Residuumsgleichung $\epsilon(t)$ wie folgt definiert ist:

$$\epsilon(t) = U_0 + \int_{T_0}^{t} [f_E(\tau, U_n^0(\tau) + f_I(\tau, U_n^0(\tau)] d\tau - U_n^0(t). \tag{2.25}$$

Daraus lässt sich analog zu der expliziten Formulierung von SDC eine direkte Iterati-

onsvorschrift ableiten:

$$\begin{aligned} U_{n,j+1}^{k+1} = U_{n,j}^{k+1} + \Delta t_j [& f_E(t_j, U_{n,j}^{k+1}) - f_E(t_j, U_{n,j}^{k}) \\ & + f_I(t_{j+1}, U_{n,j+1}^{k+1}) - f_I(t_{j+1}, U_{n,j+1}^{k})] + S_j^{j+1} f(\mathbf{t}_n, \mathbf{U}_n^k) \end{aligned} \tag{2.26}$$

Im Vergleich zum expliziten SDC ergibt sich durch das Lösen der impliziten, in der Regel nichtlinearen, Funktionsauswertungen ein erheblich höherer numerischer Aufwand. Das semi-implizite SDC-Verfahren weist jedoch, wie in [10] gezeigt wurde, gute Stabilitätseigenschaften auf, wodurch es sich auch für steife Anfangswertprobleme eignet, die häufig aus räumlich diskretisierten parabolischen Randwertproblemen resultieren.

2.2.3 Eigenschaften

Die Kehrseite der großen Flexibilität und der hohen Konvergenzordnung von SDC ist der hohe Berechnungsaufwand pro Zeitschritt. SDC-Verfahren, die ein Verfahren erster Ordnung für die Lösung der Fehlergleichung (2.9) verwenden, benötigen K Iterationen pro Zeitschritt, um eine Genauigkeit von K-ter Ordnung zu erreichen. Nach Dutt beträgt die Fehlerordnung von SDC bei der Wahl des Gauß-Lobatto-Verfahrens für die numerische Quadratur in der Gleichung (2.16) $\mathcal{O}(h^{\min[K,2m-2]})$, wobei die Fehlergleichung durch ein Verfahren erster Ordnung diskretisiert wurde. K ist dabei die Anzahl der SDC-Iterationen und m bezeichnet die Anzahl der Quadraturknoten in der Gleichung (2.16). Die maximale Konvergenzordnung wird durch die Wahl des Quadraturverfahrens bestimmt. Beim Gauß-Lobatto beträgt die Fehlerordnung $2m-2$, bei Gauß-Radau hingegen $2m-1$. Da aufgrund der dem SDC-Verfahren zugrunde liegenden numerischen Quadratur in jeder Iteration die Lösung an einer Anzahl an Zwischenschritten berechnet werden muss, die proportional zu der gewünschten Fehlerordnung ist, wächst die Anzahl an Funktionsauswertungen quadratisch mit der Ordnungszahl. Dadurch wirken die Kosten pro Zeitschritt verglichen mit anderen klassischen Methoden wie zum Beispiel Runge-Kutta-Methoden sehr hoch. Jedoch können aufgrund des aus der hohen Fehlerordnung resultierenden größeren Stabilitätsgebiets die Schrittweite für die zeitliche Diskretisierung vergrößert werden, sodass der hohe Berechnungsaufwand relativiert wird. Außerdem wurde in [25] gezeigt, dass der Berechnungsaufwand von SDC mit dem von Runge-Kutta-Methoden hoher Ordnung vergleichbar ist. Das SDC-Verfahren hat zudem den Vorteil, dass es im Gegensatz zu Runge-Kutta-Methoden eine semi-implizite Variante mit Fehlerordnungen größer als sechs gibt, wodurch es für die sehr genaue Lösung von steifen Problemen geeignet ist. Steife Probleme tauchen vornehmlich bei der Diskretisierung von zeitabhängigen partiellen Differentialgleichungen, wie die Wärmeleitungsgleichung, über die Linienmethode [30] auf und deswegen ist es in der Praxis von großem Vorteil, dass sich SDC dank der semi-impliziten Formulierung auch für steife Probleme einsetzen lässt.

In der Veröffentlichung von Dutt et. al. [10] wurde die Stabilität von SDC durch nu-

merische Auswertungen analysiert. Dabei wurden drei Versionen von SDC betrachtet, nämlich die explizite SDC-Version, die in Abschnitt 2.2.1 hergeleitet wurde, eine implizite Version, bei der die Fehlergleichung (2.9) über das implizite Euler-Verfahren diskretisiert wird und eine kombinierte implizite Version, die zwei implizite Versionen mit verschieden Iterationszahlen miteinander kombiniert. Für die explizite SDC-Version wurden schlechte Stabilitätseigenschaften nachgewiesen, die von der schlechten Stabilität des expliziten Euler-Verfahrens verursacht wird. Es stellte sich heraus, dass die implizite SDC-Version für verschiedene Kombinationen von den Parametern, Anzahl der Quadraturknoten m und Iterationszahl K, $A(\alpha)$-stabil mit einem α-Wert nahe 90° ist, jedoch nicht L-stabil ist. Um die L-Stabilität zu erreichen, wurde die kombinierte Version verwendet, die für alle Parameterkombinationen L-stabil ist. Die in Abschnitt 2.2.2 hergeleitete semi-implizite SDC-Version besitzt ebenfalls gute Stabilitätseigenschaften [25].

2.2.4 Weiterentwicklungen

Um den hohen Berechnungsaufwand pro Iteration von SDC zu umgehen, wurde mit Anleihen aus dem Bereich der *Mehrgitterverfahren* das *Multilevel-SDC-Verfahren (MLSDC)* von Speck et al. [33] entwickelt. Beim MLSDC-Verfahren wird eine Hierarchie von gröber werdenden Berechnungsgittern eingeführt. Die Berechnung der Iterationsvorschrift wird auf dem groben Gitter ausgeführt und dadurch verringert sich der Berechnungsaufwand deutlich. Für die Migration der Informationen zwischen den Gittern wird die von den nichtlinearen Mehrgitterverfahren bekannte *FAS-Korrektur (Full Approximation Scheme)* angewandt [8]. Dadurch erhalten die Näherungen auf den groben Gittern die gleiche Genauigkeit wie die Näherung auf dem feinen Gitter. Das Prinzip von MLSDC wird auch von dem *PFASST*-Algorithmus (*Parallel Full Approximation Scheme in Space and Time*) [11] verfolgt, wobei hier eine Zeitparallelisierung im Sinne von Parareal angewandt wird. PFASST kombiniert somit das MLSDC-Verfahren mit dem voll-hybriden Parareal/SDC-Algorithmus, der in Kapitel 3 thematisiert wird.

Eine weitere Weiterentwicklung von SDC mit der Motivation, den Berechnungsaufwand zu verringern, ist das *Inexact SDC-Verfahren (ISDC)* [35]. Dabei werden die Gleichungssysteme, die aufgrund der impliziten Funktionsauswertungen in (2.21) auftreten, im Gegensatz zum klassischen SDC-Verfahren nicht bis zur vollen Konvergenz gelöst. Es zeigt sich, dass die für die Konvergenz benötigte Iterationszahl steigt, jedoch nicht so stark, dass insgesamt eine Effizienzsteigerung im Vergleich zum klassischen SDC-Verfahren erreicht wird.

Der hohe Berechnungsaufwand des SDC-Verfahrens wird durch die Verbindung mit Parareal amortisiert, da die Iterationen beider Verfahren kombiniert werden, sodass die Berechnung der SDC-Iteration im Vergleich zu klassischen nicht-iterativen Verfahren wie Runge-Kutta-Methoden pro Parareal Iteration nicht so sehr ins Gewicht fällt.

3 Der hybride Parareal/SDC Algorithmus

In diesem Kapitel wird der hybride Parareal/SDC-Algorithmus thematisiert, der 2010 von Minion entwickelt wurde [25]. Die Grundidee hinter dem hybriden Parareal/SDC-Algorithmus ist die Verwebung der Parareal-Iterationen mit den SDC-Iterationen. Die Motivation hierfür ist die Verbesserung der parallelen Effizienz von Parareal, der, wie in Abschnitt 2.1.4 gezeigt wurde, unter der Grenze von $\frac{1}{2}$ liegt. Dieser Kapitel ist strukturell zweigeteilt. Im ersten Unterkapitel wird die Methodik besprochen und es werden zwei Versionen des hybriden Parareal/SDC-Algorithmus postuliert. In Abschnitt 3.1.4 wird außerdem die theoretische parallele Effizienz hergeleitet. Im zweiten Unterkapitel geht es anschließend um die Implementierung der Algorithmen im *PFASST++* Framework. Dabei werden drei Implementierungen, nämlich das klassische Parareal und die zwei hybriden Versionen, jeweils in einem Abschnitt beschrieben.

3.1 Methodik

Parareal ist ein iterativer Algorithmus, der in jeder Iteration zwei Einschrittverfahren/Propagatoren verwendet. Dabei werden fast keine Einschränkungen an diese Verfahren gemacht, außer dass es sich um Einschrittverfahren handelt und dass der grobe Propagator schneller ist als der feine Propagator. Für Parareal sind die Propagatoren *Blackbox*-Elemente, wodurch die Implementierung sehr vereinfacht wird. Das SDC-Verfahren erfüllt alle Bedingungen für die Wahl als Propagator, sodass es naheliegt SDC mit Parareal zu verwenden. Dabei gibt es verschiedene Möglichkeiten, die im folgenden besprochen werden. Im nächsten Abschnitt wird die Verwendung von SDC ohne Anpassung von Parareal besprochen. In den Abschnitten 3.1.2 und 3.1.3 werden die teil-hybride und die voll-hybride Variante von Parareal behandelt, bei denen das Blackbox-Prinzip von Parareal nicht mehr weiterverfolgt wird. In Abschnitt 3.1.4 wird die parallele Effizienz des hybriden Parareal/SDC hergeleitet. Abschließend werden in Abschnitt 3.1.5 die verschiedenen Vergröberungsstrategien, durch die Einfluss auf die parallele Effizienz genommen werden kann, besprochen.

3.1.1 SDC als klassischer Propagator

Beim klassischen Vorgehen ist keine Anpassung von Parareal notwendig. In jeder Parareal-Iteration wird seriell der grobe Propagator und parallel der feine Propagator verwendet, wobei es sich bei beiden Propagatoren in diesem Fall um die volle Ausführung des SDC-Verfahrens handelt. Das heißt, es werden mehrere SDC-Iterationen ausgeführt, bis das Residuum (2.12) kleiner ist als eine vom Anwender vorgegebene Toleranz, d.h. in der Regel bis das SDC-Verfahren konvergiert ist. Bei der Implementierung von Parareal kann in diesem Fall das SDC-Verfahren als Blackbox-Element verwendet werden, da nur ein Anfangswert U_n^k angegeben werden muss, um die Näherung U_{n+1}^k zu erhalten. Es werden zwei Konfigurationen von SDC benötigt, zum einen die Konfiguration für den groben Propagator und zum anderen die Konfiguration für den feinen Propagator. Für die Vergröberung gibt es verschiedene Möglichkeiten, die in Abschnitt 3.1.5 näher besprochen werden. Zunächst wird seriell eine Initialisierungsphase mit dem groben Propagator gestartet, sodass für die Näherung von Prozess n in der 0-ten Iteration, $\tilde{U}_{n+1}^0 := \mathbf{P}G(t_{n+1}, t_n, \mathbf{R}U_n^0)$ gilt. Diese Näherung wird an Prozess $n+1$ als neuer Anfangswert U_n^0 geschickt. In jeder weiteren Iteration, also für $k > 0$, wird zusätzlich auch der feine Propagator, also SDC mit der feinen Konfiguration, ausgeführt, sodass für die feine Näherung $\hat{U}_{n+1}^k := F(t_{n+1}, t_n, U_n^{k-1})$ gilt. Im Gegensatz zur Initialisierungsphase wird nun die über die Parareal vorschrift (2.3) berechnete Korrektur $U_{n+1}^k := \hat{U}_{n+1}^k + \tilde{U}_{n+1}^k - \tilde{U}_{n+1}^{k-1}$ an Prozess $n+1$ als neuer Anfangswert U_n^k geschickt. Die Interpolations- $\mathbf{P}$ und Restriktionsoperatoren $\mathbf{R}$ beziehen sich hierbei ausschließlich auf den räumlichen Transfer. Als Abbruchkriterium wird die Maximumsnorm der Differenz der Iterierten $\left\| U_{n+1}^k - U_{n+1}^{k-1} \right\|_\infty$ überprüft. Wenn die Norm kleiner als eine vom Anwender angegebene Toleranz ist, wird von dem jeweiligen Prozess keine weitere Parareal-Iteration mehr ausgeführt. Die in Abschnitt 2.1.4 hergeleitete parallele Effizienz gilt auch für diese Parareal-Variante.

3.1.2 Teil-hybrides Parareal/SDC

Die teil-hybride Parareal/SDC-Version unterscheidet sich vom klassischen Parareal mit SDC grundsätzlich dadurch, dass anstatt mehrerer SDC-Iterationen bis zur Konvergenz eine feste Anzahl von Iterationen pro Parareal-Iteration ausgeführt wird (i.A. nur eine). Der Parareal Algorithmus wird somit fest mit dem SDC-Algorithmus verknüpft, sodass die in der klassischen Formulierung postulierte freie Wahl der Propagatoren nicht mehr gegeben ist. Parareal behandelt die Propagatoren in der klassischen Formulierung als Blackbox-Elemente, die als Input einen Anfangswert erhalten und als Output die Näherung zum nächsten Zeitpunkt zurück geben. Beim teil-hybriden Parareal ist dieses Blackbox-Prinzip noch intakt, mit der besagten Einschränkung, dass SDC als Propagator fest vorgegeben wird. Außerdem wird das Dreiecksschema, wie es in Abbildung 2.2 dargestellt ist, verletzt. Dadurch, dass in jeder Parareal-Iteration eine SDC-Iteration

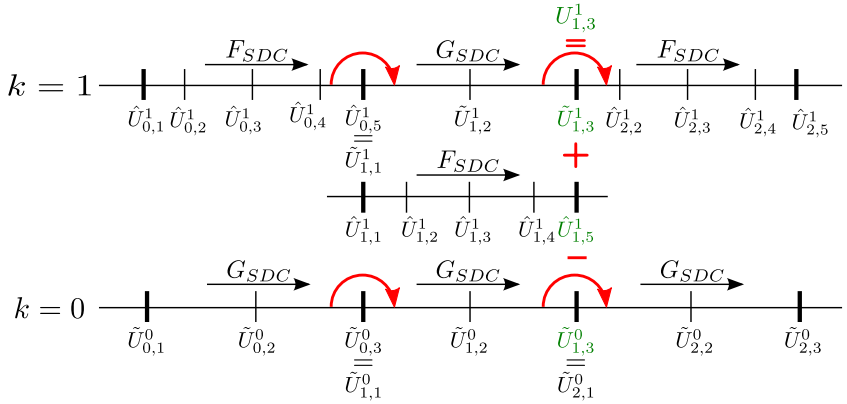

Abbildung 3.1: Graphische Darstellung der Funktionsweise des teil-hybriden Parareal/SDC.

ausgeführt wird, gilt für die Iterationszahl k von Parareal nicht mehr $k \leq n$, sodass sich kein Dreiecksschema mehr ergibt. Beim klassischen Parareal führt der erste Prozess nur eine Parareal-Iteration aus, da die Näherung des voll konvergierten feinen Propagators nach der ersten Iteration vorliegt und keine serielle Korrektur benötigt wird. Da beim hybriden Parareal/SDC-Verfahren nur eine SDC-Iteration pro Parareal-Iteration ausgeführt wird, muss auch Prozess 0 mehrere Parareal-Iterationen ausführen, bis die Konvergenz erreicht wird. Die Ausführung einer SDC-Iteration (2.21) mit der groben Konfiguration wird im folgenden mit $\tilde{U}^k_{n,j+1} = G_{SDC}(\tilde{U}^k_{n,j})$ bezeichnet. Der Knotenindex j geht hierbei von 1 bis $\tilde{J}$. Die Knoten in der numerischen Quadratur (2.16) werden dabei so gewählt, dass die Näherung $\tilde{U}^k_{n+1}$ identisch ist mit $\tilde{U}^k_{n,\tilde{J}}$. Dies ist beispielsweise für Gauß-Lobatto Knoten der Fall. Dementsprechend wird die Ausführung einer SDC-Iteration mit der feinen Konfiguration mit $\hat{U}^k_{n,j+1} = F_{SDC}(\hat{U}^k_{n,j})$ bezeichnet, wobei hier j von 1 bis $\hat{J}$ geht. Die Iterationsvorschrift von Parareal (2.3) wird auf die Näherungen an den Endknoten des feinen und groben SDC Propagators angewandt, sodass die Näherung $U^k_{n+1} = \mathbf{R}\hat{U}^k_{n,\hat{J}} + \tilde{U}^k_{n,\tilde{J}} - \tilde{U}^{k-1}_{n,\tilde{J}}$ an den nächsten Prozess als neuer Anfangswert U^k_n gesendet wird. Hierbei ist zu beachten, dass die weitergesendete Näherung U^k_{n+1} die räumliche Auflösung des groben Gitters besitzt, damit der grobe Propagator des nächsten Prozesses diesen Anfangswert sofort für die Berechnung der nächsten Näherung für die serielle Korrektur verwenden kann, ohne vorher eine räumliche Restriktion ausführen zu müssen. Dadurch ist der serielle Anteil des Algorithmus geringer, als wenn die kommunizierten Näherungen auf dem feinen Gitter aufgelöst wären. Die Prozesse P_0 und P_{N_p} führen außer in der ersten Iteration ($k = 0$) nur F_{SDC} aus, da G_{SDC} nur für die Propagation eines neuen Anfangswerts an den nächsten Prozess benötigt wird und der erste Prozess keinen neuen Anfangswert bekommt bzw. der letzte Prozess keinen Nachfolger hat. Wie beim klassischen Parareal mit SDC wird auch hier die Maximumsnorm der Differenz der Iterierten für die Abbruchbedingung verwendet. Die Funktionsweise des teil-hybriden Parareal/SDC wird in Abbildung 3.1 beispielhaft für drei Prozesse graphisch dargestellt. Die grobe SDC-Iteration wird hier mit drei Quadratur-Knoten

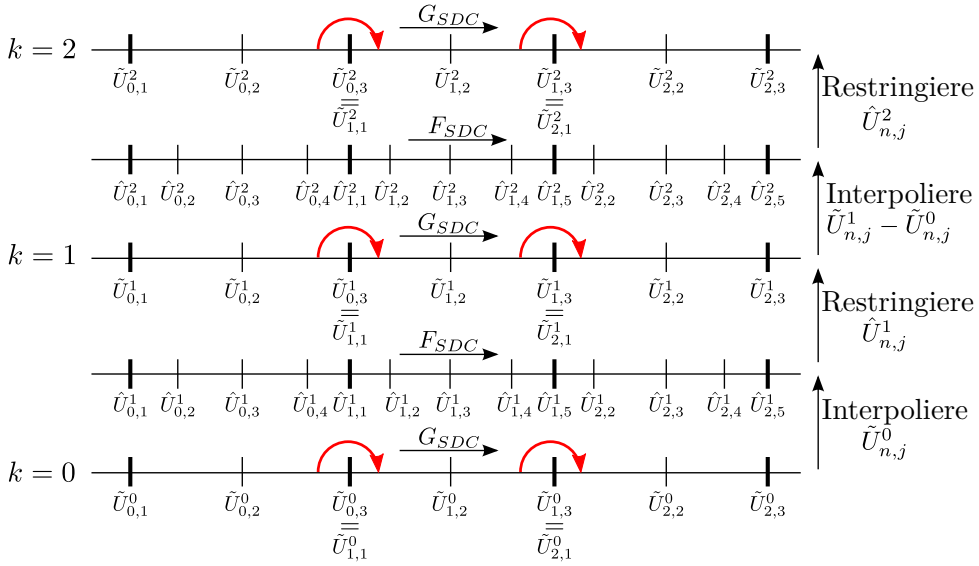

Abbildung 3.2: Graphische Darstellung der Funktionsweise des voll-hybriden Parareal/SDC.

und die feine SDC-Iteration mit fünf Knoten berechnet. Die Berechnung der seriellen Parareal-Korrektur wird gesondert farblich hervorgehoben. Die Weitersendung der neuen Anfangswerte wird durch die roten Pfeile symbolisiert.

3.1.3 Voll-hybrides Parareal/SDC

Die voll-hybride Parareal/SDC-Variante basiert auf dem teil-hybriden Parareal/SDC, wobei hier noch zusätzlich Interpolations- und Restriktionsoperatoren eingesetzt werden, um die serielle Korrektur durch die Näherungen des groben SDC Propagators auf die Näherungen des feinen SDC Propagators anzuwenden und umgekehrt die Näherungen des feinen SDC Propagators für den groben SDC Propagators zu verwenden. Nach der Initialisierungsphase mit dem groben SDC Propagator werden die Näherungen $\tilde{U}^k_{n,j}$ an den groben Zeitpunkten auf die feinen Zeitpunkte interpoliert. Diese interpolierten Werte werden anschließend in der ersten Iteration ($k = 1$) von dem feinen SDC Propagator für die Berechnung der Näherungen $\hat{U}^k_{n,j}$ verwendet. Die Näherungen des feinen SDC Propagators an den feinen Zeitpunkten werden vor der Ausführung des groben SDC Propagators auf die groben Zeitpunkte restringiert. Jede weitere Iteration ($k > 1$) läuft fast genauso ab, mit dem Unterschied, dass nun die Differenz der groben Näherungen $\tilde{U}^k_{n,j}$ der aktuellen Iteration und den groben Näherungen $\tilde{U}^{k-1}_{n,j}$ an den feinen Zeitpunkten interpoliert und auf die feinen Näherungen $\hat{U}^k_{n,j}$ aufaddiert wird. Dadurch wird die serielle Parareal-Korrektur an allen inneren Knoten berechnet, anstatt nur im letzten Knoten wie beim teil-hybriden Parareal/SDC. Die Näherungen an allen inneren Knoten werden

somit für die nächste Iteration wiederverwendet, wodurch eine Verringerung der Anzahl der benötigten Iterationen und somit auch eine höhere parallele Effizienz zu erwarten ist. Ein weiterer Unterschied ist, dass alle Prozesse in jeder Iteration sowohl den groben, als auch den feinen SDC Propagator ausführen. Es wird immer die Näherung des groben SDC Propagators am letzten Knoten, also $\tilde{U}^k_{n,\tilde{J}}$ an den nächsten Prozess gesendet. In Abbildung 3.2 wird die Funktionsweise des voll-hybriden Parareal/SDC veranschaulicht. Dabei wird die selbe Konfiguration wie in Abbildung 3.1 verwendet. Für die Abbruchbedingung wird die Maximumsnorm der Differenz der aktuellen und der letzten Näherung des feinen SDC im letzten Knoten $\left\|\hat{U}^k_{n,\hat{J}} - \hat{U}^{k-1}_{n,\hat{J}}\right\|_\infty$ berechnet.

3.1.4 Parallele Effizienz

In diesem Abschnitt wird die parallele Effizienz des hybriden Parareal/SDC wie in [25] hergeleitet. Dafür wird die Notation aus Abschnitt 2.1.4 etwas modifiziert. Die Laufzeit für die Ausführung einer groben SDC-Iteration wird mit τ_G bezeichnet. Dementsprechend wird die Laufzeit einer feinen SDC-Iteration mit τ_F bezeichnet. Für die Berechnung des Speedups wird die parallele Laufzeit mit der Laufzeit des seriell ausgeführten feinen SDC-Verfahrens verglichen. Dabei werden für die gewünschte Genauigkeit M SDC-Iterationen benötigt. Dadurch ergibt sich für das seriell ausgeführte feine SDC-Verfahren bei N_p Zeitschritten eine Laufzeit von $N_pM\tau_F$. Die parallele Laufzeit beträgt, wie in Abbildung 3.3 ersichtlich wird, $N_p\tau_G + K(\tau_G + \tau_F)$. Der Speedup des hybriden Parareal/SDC ist somit:

$$S = \frac{N_pM\tau_F}{N_p\tau_G + K(\tau_G + \tau_F)} = \frac{M}{\frac{\tau_G}{\tau_F} + \frac{K}{N_p}(\frac{\tau_G}{\tau_F} + 1)}. \tag{3.1}$$

Durch die Einführung des Faktors $\alpha = \frac{\tau_G}{\tau_F}$ erhält man:

$$S = \frac{M}{\alpha + \frac{K}{N_p}(\alpha + 1)}. \tag{3.2}$$

Der theoretisch maximal mögliche Speedup ist somit genau das M-fache des theoretisch maximalen Speedups des klassischen Parareal (2.5). Für die Effizienz des hybriden Parareal/SDC gilt:

$$E = \frac{M}{\alpha \cdot N_p + K(\alpha + 1)}. \tag{3.3}$$

Damit ist auch die parallele Effizienz des hybriden Parareal/SDC genau das M-fache der parallelen Effizienz des klassischen Parareal. Die Effizienzsteigerung ist dadurch zu erklären, dass die SDC-Iterationen mit den Parareal-Iterationen verknüpft werden, sodass der Berechnungsaufwand pro Parareal-Iteration im Vergleich zum klassischen Parareal verringert werden. Der Faktor α spielt, wie beim klassischen Parareal eine bedeutende Rolle für die parallele Effizienz. Je kleiner dieser Faktor ist, also je schneller der grobe

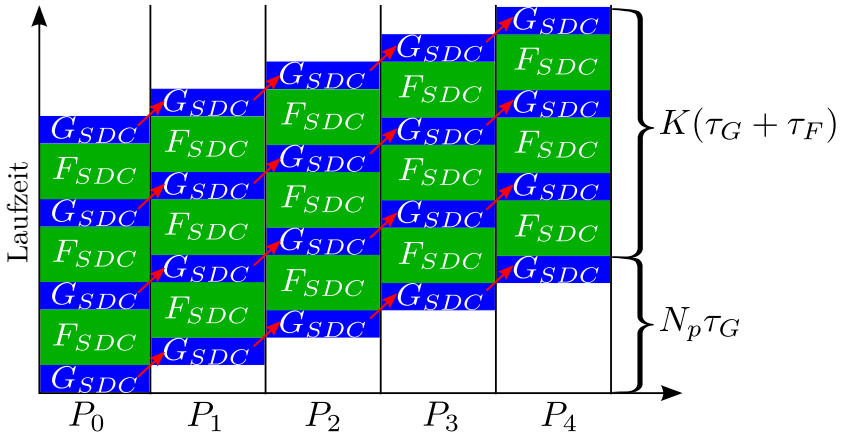

Abbildung 3.3: Graphische Darstellung der Laufzeit des hybriden Parareal/SDC

SDC im Vergleich zum feinen SDC ist, desto höher kann die parallele Effizienz sein. Eine zu starke Vergröberung kann jedoch die Anzahl der Parareal-Iterationen K, bis die gewünschte Genauigkeit erreicht wird, in die Höhe treiben und sich wiederum negativ auf die parallele Effizienz auswirken. Die Möglichkeiten der Vergröberung bei Parareal in Verbindung mit SDC werden im nächsten Abschnitt besprochen.

3.1.5 Vergröberungsstrategien

In der klassischen Herangehensweise bei Parareal wird die Vergröberung, sprich die Verringerung der Laufzeit des groben Propagators im Vergleich zur Laufzeit des feinen Propagators, über die Verwendung von unterschiedlichen Schrittweiten innerhalb des selben Verfahrens realisiert. Dabei ist die Schrittweite Δt des groben Propagators viel größer als die Schrittweite des feinen Propagators δt. Bei Parareal in Verbindung mit SDC jedoch wird die Vergröberung etwas anders behandelt.

Grundsätzlich kann bei SDC zwischen zwei Arten der Vergröberung unterschieden werden. Die räumliche Vergröberung geschieht durch die Verwendung von verschieden aufgelösten räumlichen Berechnungsgittern. Bei nicht gitterbasierter Diskretisierung kann die Anzahl der räumlichen Freiheitsgrade auch auf andere Weise verringert werden. Bei räumlicher Vergröberung ist die Anzahl der räumlichen Freiheitsgrade für das grobe SDC $\tilde{N}_x$ geringer als die Anzahl der Freiheitsgrade des feinen SDC $\hat{N}_x$. Beispielsweise verringert sich die Anzahl der Freiheitsgrade für ein dreidimensionales Problem bei einer Reduzierung der räumlichen Auflösung um einen Faktor zwei schon um das achtfache [25].

Die andere Möglichkeit der Vergröberung ist die Verwendung einer geringeren Anzahl an Quadraturknoten $\tilde{J}$ für das grobe SDC, als die Anzahl der Quadraturknoten $\hat{J}$ für das

feine SDC. Dadurch verringert sich die Anzahl der Funktionsauswertungen des groben SDC, sodass der Berechnungsaufwand pro Iteration sinkt. Die Konvergenzordnung wird jedoch auch reduziert, sodass die Näherungen des groben SDC ungenauer sind als die des feinen SDC. Die räumliche Vergröberung kann mit der Verringerung der Knotenanzahl kombiniert werden, wodurch der Berechnungsaufwand des groben SDC noch stärker reduziert werden kann. Die hier besprochenen Vergröberungsstrategien werden in Kapitel 4 in Abschnitt 4.3 numerisch miteinander verglichen.

3.2 Implementierung im PFASST++ Framework

In diesem Abschnitt geht es um die Implementierung der oben beschriebenen Parareal/SDC-Varianten im *PFASST++* Framework in der Version 0.5.0 [19]. Es handelt sich dabei um die C++ Implementierung des in Abschnitt 2.2.4 angesprochenen PFASST-Algorithmus [11]. Zunächst werden die verschiedenen Komponenten des Frameworks besprochen, von denen in den Implementierungen Gebrauch gemacht wird. Anschließend werden die verschiedenen Implementierungen in jeweils separaten Abschnitten kommentiert. Zu jeder Implementierung wird ein Pseudocode angegeben. Abschließend geht es in Abschnitt 3.2.5 um die Auswertung der numerischen Ergebnisse über eigens dafür erstellte Python-Skripte.

Die Prozesse werden zyklisch eingesetzt. Das bedeutet, dass mehrere Parareal-Blöcke ausgeführt werden, bis der zu berechnende endgültige Zeitpunkt T erreicht wird. Die Anzahl der Blöcke N_b wird folgendermaßen berechnet:

$$N_b = \left\lceil \frac{(T - T_0)/\Delta t}{N_p} \right\rceil. \tag{3.4}$$

Dabei bezeichnet Δt die vom Anwender gewählte Schrittweite, die von einem Prozess bearbeitet wird und N_p ist die Anzahl der Prozesse. In Tabelle 3.1 sind alle Eingabe-Parameter, die von den Implementierungen benötigt werden, aufgeführt.

3.2.1 Aufbau von PFASST++

PFASST++ ist strukturell problemunabhängig. Das heißt, das Framework kann für verschiedene Probleme verwendet werden, in dem eigene Implementierungen der vorgegebenen Schnittstellen verwendet werden. Die wichtigsten Schnittstellen sind die *Controller*- und die *Sweeper*-Schnittstelle. Ein Controller steuert den Berechnungsablauf, das Fortschreiten in der Zeit und den Abbruch der Iteration bei Erreichen der Abbruchbedingung. Der Controller setzt die Zeitschritte für jeden Prozess und steuert die Kommunikation zwischen den Prozessen. Das Framework enthält SDC als Controller-Implementierung.

Parametername	**Beschreibung**
T	Endzeitpunkt
Δt	Zeitschritt
N_p	Anzahl der Prozesse
$\tilde{J}$	Anzahl der Quadraturknoten für das grobe SDC
$\hat{J}$	Anzahl der Quadraturknoten für das feine SDC
$\tilde{N}_x$	Anzahl der räumlichen Freiheitsgrade für das grobe SDC
$\hat{N}_x$	Anzahl der räumlichen Freiheitsgrade für das feine SDC
N_k	Maximale Anzahl von Parareal-Iterationen
ϵ	Toleranzwert für die Abbruchbedingung

Tabelle 3.1: Eingabeparameter

Der SDC-Controller kann über die Parameter, Anzahl der räumlichen Freiheitsgrade $\hat{N}_x$ und Anzahl der Quadraturknoten $\hat{J}$, konfiguriert werden. Dabei wird pro Iteration ein *Sweep* ausgeführt. Ein Sweep ist die Ausführung der problemspezifischen *Sweeper*-Implementierung. Darin werden die Funktionsauswertungen und die Berechnung der numerischen Quadratur für die SDC-Iteration (2.21) ausgeführt. Das Framework enthält zudem die Berechnungsfunktionen für die Quadratur-Matrizen für verschiedene Knoten-Typen. Eine weitere wichtige Schnittstelle ist die *Transfer*-Schnittstelle, über die Näherungen im Raum und Zeit interpoliert und restringiert werden können.

3.2.2 Implementierung des klassischen Parareal

In diesem Abschnitt wird die Implementierung des klassischen Parareal beschrieben. Der Pseudocode zu der Implementierung wird in Algorithmus 1 aufgeführt. Das klassische Parareal in Verbindung mit SDC verwendet die SDC-Controller-Implementierung von PFASST++. Es werden zwei SDC-Controller-Instanzen angelegt, zum einen die Instanz des feinen SDC, mit der Konfiguration $\hat{N}_x$ und $\hat{J}$, und zum anderen die des groben SDC, mit der Konfiguration $\tilde{N}_x$ und $\tilde{J}$. Diese Instanzen sind die Entsprechungen zu den Propagatoren, die von Parareal verwendet werden. Parareal hat durch diese Art der Implementierung nur Zugriff auf die Schnittstellen-Funktionen der SDC-Controller-Instanzen. Dies entspricht dem Blackbox-Prinzip von Parareal, da Parareal formal keine Kenntnisse der internen Berechnungen der Propagatoren benötigt. Pro Parareal-Iteration werden die SDC-Controller durch den Aufruf der `run`-Funktion gestartet. Die SDC-Controller führen dann mehrere SDC-Iterationen aus, bis das Residuum unter die Toleranzgrenze fällt. Die `run`-Funktion terminiert erst nach der Beendigung der letzten SDC-Iteration. Die neuen Anfangswerte werden über den Zeiger `get_start_state()` gesetzt. Die räumliche Interpolation der Näherungen des groben SDC-Controllers wird über eine *Transfer*-Instanz ausgeführt.

3.2.3 Implementierung des teil-hybriden Parareal/SDC

Im Unterschied zum klassischen Parareal ist das teil-hybride Parareal/SDC als eigener Controller implementiert. Deswegen werden keine SDC-Controller-Instanzen verwendet, sondern *Sweeper*-Instanzen mit den jeweiligen Konfigurationen. In jeder Parareal-Iteration wird jeweils einmal die `sweep`-Funktion des feinen und des groben *Sweepers* aufgeführt. Die `sweep`-Funktion berechnet die SDC-Iteration. Über die Schnittstellenfunktion `get_end_state()` wird die Näherung im letzten Knoten von den Sweepern abgefragt, um die serielle Parareal-Korrektur, wie in Abschnitt 3.1.2 beschrieben, zu berechnen. Wie beim klassischen Parareal wird auch hier ausschließlich räumliche Interpolation bzw. Restriktion verwendet. Die im Endknoten korrigierte Näherung wird an den nächsten Prozess gesendet, wo sie als neuer Anfangswert für den groben und feinen *Sweeper* über den Pointer `get_start_state()` gesetzt wird. Der Pseudocode des teil-hybriden Parareal/SDC ist in Algorithmus 2 angegeben.

3.2.4 Implementierung des voll-hybriden Parareal/SDC

Die Implementierung des voll-hybriden Parareal/SDC verwendet neben der räumlichen auch die zeitliche Interpolation und Restriktion, da die Parareal-Korrektur im Gegensatz zum teil-hybriden Parareal/SDC nicht nur im Endknoten sondern in allen Zwischenknoten berechnet wird. Die *Transfer*-Schnittstelle im PFASST++ Framework bietet bereits die Funktionalität der zeitlichen und räumlichen Interpolation und Restriktion in jeweils einem Funktionsaufruf. Die Weitergabe von den neuen Anfangswerten läuft ausschließlich auf der Stufe des groben SDC ab. In jeder Parareal-Iteration wird vor der Ausführung der `sweep`-Funktion des feinen *Sweepers* die `interpolate`-Funktion der `Transfer`-Instanz aufgerufen, wodurch die Parareal-Korrektur auf die feinen Näherungen angewandt wird. Nach dem `sweep`-Aufruf werden die Ergebnisse des feinen *Sweepers* auf den groben *Sweeper* über die `restrict`-Funktion der *Transfer*-Instanz restringiert. Dadurch kann auf der groben Stufe mit den genaueren Werten gerechnet werden. Die Näherung im letzten Knoten des groben *Sweepers* wird weitergesendet. Der Pseudocode des voll-hybriden Parareal/SDC ist in Algorithmus 3 aufgeführt.

3.2.5 Python-Skripte für das Postprocessing

Um die numerischen Ergebnisse der Implementierungen auswerten zu können, ist es hilfreich, aus den Berechnungen aussagekräftige Plots zu erstellen. Für das in Kapitel 4 verwendete Beispielproblem, das Advektions-Diffusions-Problem, kann die Lösung analytisch berechnet werden, sodass der absolute Fehler der numerischen Näherungen direkt angegeben werden kann. Dadurch ist es möglich, die Konvergenz der Implementierungen zu überprüfen. Zu jedem Zeitschritt n und zu jeder Iteration k muss daher der Fehler

bekannt sein. Jeder Prozess legt bei jeder Ausführung eine Log-Datei an, in die unter anderem genau diese Information gespeichert wird. Um die Fehler-Plots erstellen zu können, wurden im Rahmen dieser Arbeit Python-Skripte geschrieben, die diese Log-Dateien einlesen und in einen eigenen Datentyp transferieren können. Dieser Datentyp kann über das *pickle*-Modul in eine eigene Datei gespeichert und für verschiedene Plots ausgelesen werden. In dem Datentyp wird in einer `Dictionary`-Instanz zu jedem n und k sowohl der Fehler als auch das Residuum und die gemessenen Laufzeiten gespeichert

Algorithm 1 Implementierung des klassischen Parareal

for $i := 0, \dots, N_b - 1$ **do** ▷ Schleife über die Zeitblöcke
  $n := N_p \cdot j + rank$
  $initial := rank = 0$ & $i = 0$
  if $n \cdot \Delta t > T$ **then**
    break
  end if
  for $k := 0, \dots, N_k - 1$ **do** ▷ Schleife über die Parareal-Iterationen
    if *done* oder $k > 0$ & $rank < k - 1$ **then**
      break
    end if
    if $k > 0$ **then**
      $\hat{U}^k_{n+1} := F(t_{n+1}, t_n, U^{k-1}_n)$ ▷ Feiner Propagator
    end if
    if $rank \geq k$ **then**
      if !*initial* & !*precdone* **then**
        erhalte U^k_n von Prozess $rank - 1$
        if $rank > 0$ **then**
          erhalte *done* von Prozess $rank - 1$ als *precdone*
        end if
      else if !*initial* **then**
        done :=Wahr
      end if
      $\tilde{U}^k_{n+1} := \mathbf{P}G(t_{n+1}, t_n, \mathbf{R}U^k_n)$ ▷ Grober Propagator
      if $k = 0$ **then**
        $U^k_{n+1} := \tilde{U}^k_{n+1}$
      else
        $U^k_{n+1} := \hat{U}^k_{n+1} + \tilde{U}^k_{n+1} - \tilde{U}^{k-1}_{n+1}$ ▷ Korrektur
      end if
    else
      $U^k_{n+1} := \hat{U}^k_{n+1}$
    end if
    if $k > 0$ & !*done* **then**
      $done := \left\| U^k_{n+1} - U^{k-1}_{n+1} \right\|_\infty < \epsilon$
    end if
    if $rank < N_p - 1$ & $(n+1) \cdot \Delta t \leq T$ **then**
      sende U^k_{n+1}und *done* an Prozess $rank + 1$
    end if
  end for
  if $j < N_b - 1$ & $rank = N_p - 1$ **then**
    sende U^k_{n+1} an Prozess 0
  end if
end for

Algorithm 2 Implementierung des teil-hybriden Parareal/SDC

for $i := 0, \ldots, N_b - 1$ **do** ▷ Schleife über die Zeitblöcke
  $n := N_p \cdot j + rank$
  if $n \cdot \Delta t > T$ **then**
    break
  end if
  for $k := 0, \ldots, N_k - 1$ **do** ▷ Schleife über die Parareal-Iterationen
    if *done* **then**
      break
    end if
    if $k > 0$ **then**
      if $k = 1$ **then**
        Interpoliere $\tilde{U}^0_{n,j}, j = 1, \ldots, \tilde{J}$ zu $\hat{U}^1_{n,j}, j = 1, \ldots, \hat{J}$
      end if
      $\hat{U}^k_{n,j+1} = F_{SDC}(\hat{U}^k_{n,j}); \quad j = 1, \ldots, J-1$ ▷ Feine SDC-Iteration
      $done := \left\| \hat{U}^k_{n,\hat{J}} - \hat{U}^{k-1}_{n,\hat{J}} \right\|_\infty < \epsilon$
    end if
    $recv :=$!$precdone$ & ($rank > 0$ oder ($i > 0$ & $k = 0$))
    if $recv$ **then** ▷ Erhalte Anfangswert
      Erhalte $\tilde{U}^k_{n,1}$ von Prozess $rank - 1$
      Interpoliere $\tilde{U}^k_{n,1}$ zu $\hat{U}^k_{n,1}$
      if $rank > 0$ **then**
        erhalte *done* von Prozess $rank - 1$ als *precdone*
      end if
    end if
    $doCoarse :=$ $k = 0$ oder ($recv$ & $rank > 0$ & $rank < N_p - 1$)
    if $doCoarse$ **then**
      $\tilde{U}^k_{n,j+1} = G_{SDC}(\tilde{U}^k_{n,j}); \quad j = 1, \ldots, \tilde{J}-1$ ▷ Grobe SDC-Iteration
    end if
    if $rank < N_p - 1$ **then**
      if $k = 0$ **then**
        Sende $\tilde{U}^k_{n,\tilde{J}}$ an Prozess $rank + 1$
      else
        if $doCoarse$ **then**
          Sende $\mathbf{R}\hat{U}^k_{n,\hat{J}} + \tilde{U}^k_{n,\tilde{J}} - \tilde{U}^{k-1}_{n,\tilde{J}}$ an Prozess $rank + 1$
        else
          Sende $\mathbf{R}\hat{U}^k_{n,\hat{J}}$ an Prozess $rank + 1$
        end if
        Sende *done* an Prozess $rank + 1$
      end if
    end if
  end for
  if $j < N_b - 1$ & $rank = N_p - 1$ **then**
    sende $\mathbf{R}\hat{U}^k_{n,\hat{J}}$ an Prozess 0
  end if
end for

Algorithm 3 Implementierung des voll-hybriden Parareal/SDC

for $i := 0, \dots, N_b - 1$ **do** ▷ Schleife über die Zeitblöcke
  $n := N_p \cdot j + rank$
  if $n \cdot \Delta t > T$ **then**
    break
  end if
  for $k := 0, \dots, N_k - 1$ **do** ▷ Schleife über die Parareal-Iterationen
    if *done* **then**
      break
    end if
    if $k > 0$ **then**
      if $k = 1$ **then**
        Interpoliere $\tilde{U}^0_{n,j}, j = 1, \dots, \tilde{J}$ zu $\hat{U}^1_{n,j}, j = 1, \dots, \hat{J}$
      else
        Interpoliere $\tilde{U}^k_{n,j} - \tilde{U}^{k-1}_{n,j}, j = 1, \dots, \tilde{J}$ zu $\hat{U}^k_{n,j}, j = 1, \dots, \hat{J}$
      end if
      $\hat{U}^k_{n,j+1} = F_{SDC}(\hat{U}^k_{n,j}); \quad j = 1, \dots, J-1$ ▷ Feine SDC-Iteration
      $done := \left\| \hat{U}^k_{n,\hat{J}} - \hat{U}^{k-1}_{n,\hat{J}} \right\|_\infty < \epsilon$
      Restringiere $\hat{U}^k_{n,j}$ an feinen Zeitpunkten zu $\tilde{U}^k_{n,j}$ an groben Zeitpunkten
    end if
    if !*precdone* & ($rank > 0$ oder ($i > 0$ & $k = 0$)) **then** ▷ Erhalte Anfangswert
      Erhalte $\tilde{U}^k_{n,1}$ von Prozess $rank - 1$
      if $rank > 0$ **then**
        erhalte *done* von Prozess $rank - 1$ als *precdone*
      end if
    end if
    $\tilde{U}^k_{n,j+1} = G_{SDC}(\tilde{U}^k_{n,j}); \quad j = 1, \dots, \tilde{J}-1$ ▷ Grobe SDC-Iteration
    if $rank < N_p - 1$ **then**
      Sende $\tilde{U}^k_{n,\tilde{J}}$ an Prozess $rank + 1$
      if $k > 0$ **then**
        Sende *done* an Prozess $rank + 1$
      end if
    end if
  end for
  if $j < N_b - 1$ & $rank = N_p - 1$ **then**
    sende $\mathbf{R}\hat{U}^k_{n,\hat{J}}$ an Prozess 0
  end if
end for

4 Numerische Ergebnisse

In diesem Kapitel werden die Implementierungen der drei im vorherigen Kapitel besprochenen Parareal-Varianten anhand des eindimensionalen Advektions-Diffusions-Problems getestet. Für das Beispielproblem kann die Lösung analytisch exakt berechnet werden, sodass die Fehler der numerischen Näherungen bekannt sind. Dadurch ist es möglich, die Konvergenz der Implementierungen zu überprüfen. Die Konvergenztests werden in Abschnitt 4.2 besprochen. Vorher wird das Advektions-Diffusions-Problem zum besseren Verständnis thematisiert. Für die Speedup-Messungen, die in Abschnitt 4.3 besprochen werden, kam das Supercomputer System *JUQUEEN* am Forschungszentrum Jülich [37] zum Einsatz. Dabei wurden bis zu 64 Kerne eingesetzt. Die gemessenen Speedups werden mit den theoretischen Speedups, die in den Abschnitten 2.1.4 und 3.1.4 hergeleitet wurden, verglichen. Abschließend werden die in Abschnitt 3.1.5 besprochenen Vergröberungsstrategien bezüglich des Einflusses auf den Speedup untersucht und untereinander verglichen.

4.1 Das Advektions-Diffusions-Problem

In diesem Abschnitt wird das eindimensionale Advektions-Diffusions-Problem definiert und die analytische Lösung wird für das Anfangswertproblem, das in den numerischen Tests gelöst wird, angegeben. Die Advektions-Diffusions-Gleichung ist eine parabolische partielle Differentialgleichung und ist für eine Raumdimension folgendermaßen definiert:

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} - v \frac{\partial u}{\partial x}, \qquad u : \mathbb{R} \times \mathbb{R} \to \mathbb{C}. \tag{4.1}$$

Dabei ist ν der *Diffusionskoeffizient* und v die *Transportgeschwindigkeit*. Die Gleichung beschreibt eine Kombination eines Diffusionsprozesses mit einem Advektions- bzw. Transportprozess. Die Gleichung ähnelt dadurch der Wärmeleitungsgleichung und unterscheidet sich von ihr nur durch den Advektionsterm $-v\frac{\partial u}{\partial x}$. Jede Lösung $u(t, x)$ der Wärmeleitungsgleichung ist durch die Verschiebung $u(t, x - v \cdot (t - T_0))$ mit $t \geq T_0$ auch eine Lösung der Advektions-Diffusions-Gleichung. Die unbekannte Größe u beschreibt beispielsweise die Teilchenkonzentration am Ort x und zur Zeit t oder andere physikalische Größen, wie die Masse oder Temperatur. Für die numerischen Berechnungen in diesem Kapitel wird die Advektions-Diffusions-Gleichung (4.1) mit periodischen Randbedingungen und

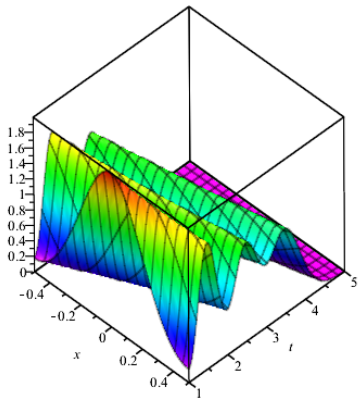

Abbildung 4.1: Darstellung der Lösung des für die numerischen Tests verwendeten Anfangswertproblems.

mit dem Anfangswert

$$u_0(x) = u(t = T_0, x) = \sum_{i=-2}^{2} \Phi(T_0, x + i), \qquad T_0 = 1, \qquad x \in \mathbb{R} \tag{4.2}$$

gelöst. Dabei bezeichnet Φ die Fundamentallösung der Wärmeleitungsgleichung

$$\Phi(t, x) = \frac{1}{\sqrt{4\pi\nu t}} \exp\left(-\frac{x^2}{4\nu t}\right). \tag{4.3}$$

Die Fundamentallösung entspricht der Gauß'schen Normalverteilung mit der Varianz $\sigma^2 = 2\nu t$. Der Anfangswert kann somit als fünf sich überlagernde Gauß-Kurven angesehen werden. Durch die Advektion bzw. Transport mit Geschwindigkeit v ist die Lösung der Advektions-Diffusions-Gleichung $u(t, x) = u_0(x - t \cdot v)$. Die analytische Lösung des Anfangswertproblems mit Anfangswert (4.2) ist somit:

$$u(t, x) = \sum_{i=-2}^{2} \Phi(x + i - (t - 1) \cdot v, t), \qquad t \geq 1, \qquad x \in \mathbb{R} \tag{4.4}$$

Abbildung 4.1 stellt die Lösung der Advektions-Diffusions-Gleichung mit dem Anfangswert (4.2) und den Parametern $\nu = 0.02$ und $v = 1$ für den Bereich $x \in [-\frac{1}{2}, \frac{1}{2}]$ als Oberflächenplot dar. Man erkennt die Abschwächung der Amplituden durch die Diffusion und die Verschiebung derselben in positiver x-Richtung durch die Advektion. Für die numerische Lösung des Problems wird das in Abschnitt 2.2.2 hergeleitete semi-implizite SDC-Verfahren eingesetzt, wobei der Advektionsterm explizit und der Diffusionsterm implizit behandelt werden. Die impliziten Funktionsauswertungen, die aus der Diskretisierung über die Linienmethode resultieren, werden über die *Fast-Fourier-Transformation (FFT)* im Spektralraum gelöst.

Vergröberung	$\hat{J}$	$\tilde{J}$	$\hat{N}_x$	$\tilde{N}_x$
nur räumlich	5	5	128	64
	3	3	128	64
kombiniert	5	3	128	64
	3	2	128	64

Tabelle 4.1: Konfigurationsparameter für die zwei Vergröberungsstrategien für die Konvergenztests.

4.2 Konvergenztests

Um die Konvergenz der Implementierungen zu testen, wurden zwei Konfigurationen gewählt. Zum einen wurde nur räumliche Vergröberung eingesetzt und zum anderen wurde sowohl räumliche als auch zeitliche Vergröberung verwendet. Die Konfigurationsparameter für die Konvergenztests sind in Tabelle 4.1 aufgelistet. Darin sind für die beiden angesprochenen Vergröberungsstrategien jeweils zwei Konfigurationen angegeben. Dabei werden die Anzahl der Knoten des feinen SDC Propagators $\hat{J}$ und des groben SDC Propagators $\tilde{J}$ variiert. Bei der ausschließlich räumlichen Vergröberung ist die Anzahl der Knoten für beide Propagatoren identisch, es gilt also $\hat{J} = \tilde{J}$. Bei der kombinierten Vergröberung wird die Anzahl der Knoten des feinen SDC Propagators $\hat{J}$ variiert und $\tilde{J}$ wird folgendermaßen bestimmt:

$$\tilde{J} = \frac{\hat{J} + 1}{2}. \tag{4.5}$$

Alle weiteren Eingabeparameter sind für alle Berechnungen identisch. Der Endzeitpunkt T ist 0.2 und die Toleranz für die Abbruchbedingung ϵ ist 10^{-14}. Die Konvergenz wird gemessen, in dem die Schrittweite Δt folgendermaßen mehrmals halbiert wird:

$$\Delta t = \frac{T - T_0}{2^i}, \qquad i = 1, \ldots, 5. \tag{4.6}$$

Der Fehler der Näherung im letzten Zeitpunkt T in der letzten Iteration K wird über die Maximumsnorm $\left\| u(T,x) - U_{N_p}^K \right\|_\infty$ ermittelt und logarithmisch in einem Diagramm auf die in Gleichung (4.6) berechneten Schrittweiten aufgetragen. Die Steigung der so entstandenen Kurve entspricht genau der Konvergenzordnung des Verfahrens. Da der Parareal-Algorithmus gegen den feinen Propagator konvergiert und der feine Propagator in den Tests das SDC-Verfahren ist, wird erwartet, dass der Fehler sich genauso wie der des feinen SDC-Verfahrens verhält. Die Konvergenzordnung von SDC entspricht wiederum der Konvergenzordnung des verwendeten Quadraturverfahrens, wie in Abschnitt 2.2.3 besprochen wurde. Für die in diesem Kapitel dargelegten numerischen Tests wurde die Gauß-Lobatto-Quadratur verwendet, die eine Fehlerordnung von $p = 2m - 2$ hat, wobei m die Anzahl der Quadraturknoten ist. Für die in Tabelle 4.1 aufgelisteten

Konfigurationsparameter können die erwarteten Fehlerordnungen somit direkt von der Anzahl der Quadraturknoten des feinen SDC Propagators $\hat{J}$ abgeleitet werden. Für die Parameterwerte $\hat{J} = 5$ und $\hat{J} = 3$ sind demzufolge die erwarteten Fehlerordnungen $p = 2 \cdot 5 - 2 = 8$ bzw. $p = 2 \cdot 3 - 2 = 4$. Das Ziel der Konvergenztests ist kurzum die Überprüfung, ob sich die Fehler der in Kapitel 3 besprochenen Implementierungen der drei Parareal-Versionen wie der Fehler des feinen SDC-Verfahrens verhalten, also ob die Implementierungen konvergieren.

In Abbildung 4.2 werden die Fehler der drei Implementierungen, also klassisches, teilhybrides- und voll-hybrides-Parareal/SDC, für die rein räumliche Vergröberung gezeigt. Die Abbildung zeigt die Fehler der drei Parareal-Implementierungen jeweils für die Parameterwerte $\hat{J} = \tilde{J} = 5$ und $\hat{J} = \tilde{J} = 3$. Außerdem werden die Fehler des feinen und des groben SDC abgebildet. Alle Kurven zu den Verfahren, für die der Parameterwert $\hat{J} = 5$ verwendet wurde, sind farblich rot dargestellt und alle Kurven zu den Verfahren, für die der Parameterwert $\hat{J} = 3$ verwendet wurde, sind farblich grün dargestellt. Darüber hinaus sind zwei gestrichelte Linien eingezeichnet, deren Steigung genau den erwarteten Fehlerordnungen $p = 8$ bzw. $p = 4$ entsprechen. Es ist deutlich sichtbar, dass die Fehler der drei Parareal-Implementierungen stark korrelieren, sodass die Kurven kaum voneinander zu unterscheiden sind. Da die Fehlerordnungen des groben und des feinen SDC aufgrund der selben Anzahl der Quadraturknoten übereinstimmen, decken sich die Fehler der beiden Verfahren. Damit konnte ein Nachweis der Konvergenz der drei Parareal-Implementierungen für die rein räumliche Vergröberung erbracht werden.

Für die Abbildung 4.3 wurden die Parameterwerte der kombinierten Vergröberung verwendet. Die Anzahl der Quadraturknoten für das grobe SDC ergibt sich durch die Gleichung (4.5). Dadurch unterscheiden sich die Fehlerordnungen des groben und des feinen SDC. Die Fehlerordnungen des feinen SDC stimmen mit den vorher berechneten Fehlerordnungen überein, sodass die Kurve des feinen SDC Propagator identisch ist mit derselben Kurve aus Abbildung 4.2. Die Fehlerordnung des groben SDC beträgt für $\tilde{J} = (5 + 1)/2 = 3$ Knoten 4 und für $\tilde{J} = (3 + 1)/2 = 2$ Knoten 2. Durch die unterschiedlichen Fehlerordnungen der Propagatoren ergeben sich die drei korrelierende Kurvenscharen in der Abbildung 4.3. Die farbliche Kennzeichnung ist wie in Abbildung 4.2, wobei hier die Kurve des groben SDC mit 2 Quadraturknoten blau dargestellt ist. Ein wesentlicher Unterschied zu Abbildung 4.2 ist, dass die Kurven des voll-hybriden Parareal/SDC mit den Kurven des groben SDC korrelieren. Das bedeutet, dass der voll-hybride Parareal/SDC gegen den groben Propagator konvergiert, anstatt gegen den feinen, wie eigentlich zu erwarten wäre. Dies ist bei den anderen Parareal-Versionen nicht der Fall. Das eigenartige Konvergenzverhalten des voll-hybriden Parareal/SDC kann dadurch erklärt werden, dass durch die Interpolation der Knotenwerte $\tilde{U}_{n,j}^{k}$ des groben SDC für die Berechnung der Knotenwerte des feinen SDC $\hat{U}_{n,j}^{k+1}$, die Konvergenzordnung des feinen SDC „verloren“ geht. Der *PFASST*-Algorithmus verwendet als Weiterentwicklung des voll-hybriden Parareal/SDC die *FAS-Korrektur* in Raum und Zeit, um das „Konvergenzproblem“ zu lösen, worauf mehr in Kapitel 5 eingegangen wird.

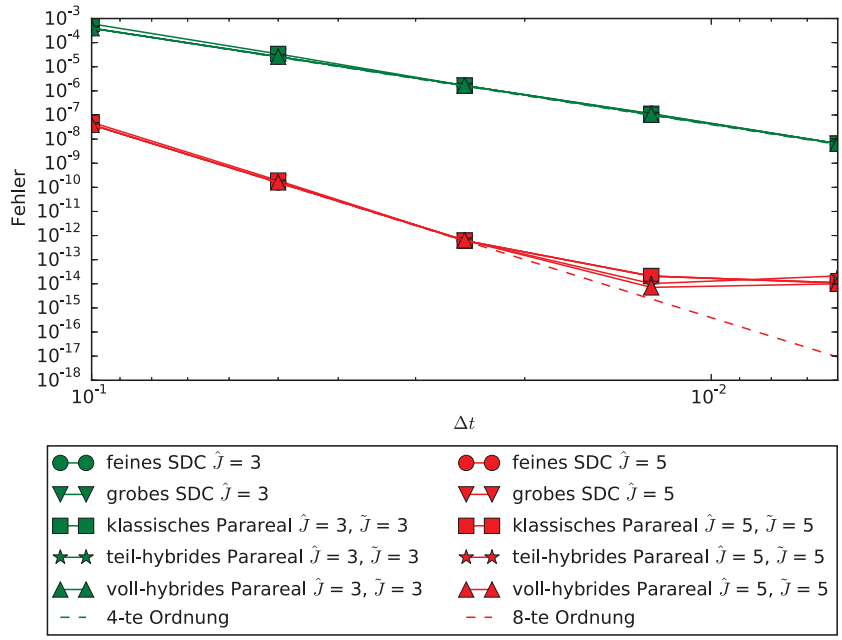

Abbildung 4.2: Konvergenz-Plot für die drei Parareal-Implementierungen mit der rein räumlichem Vergröberung.

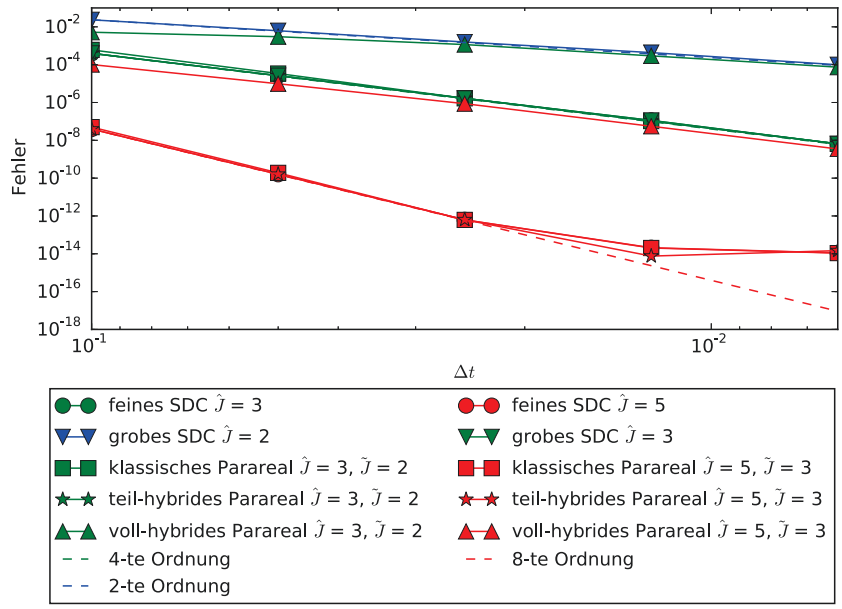

Abbildung 4.3: Konvergenz-Plot für die drei Parareal-Implementierungen mit der kombinierten Vergröberung.

Vergröberung	$\hat{J}$	$\tilde{J}$	$\hat{N}_x$	$\tilde{N}_x$
nur räumlich	5	5	4096	2048
nur zeitlich	5	3	4096	4096
kombiniert	5	3	4096	2048

Tabelle 4.2: Konfigurationsparameter für die Speeduptests.

4.3 Speeduptests

In diesem Abschnitt werden die Speeduptests besprochen. Die verwendeten Konfigurationsparameter sind in Tabelle 4.2 aufgelistet. Für die Speeduptests wurden die drei Parareal-Implementierungen mit 4, 8, 16 und 64 Kernen auf dem Supercomputer System *JUQUEEN* ausgeführt. Das in Abschnitt 4.1 definierte Anfangswertproblem mit dem Anfangswert zum Zeitpunkt $T_0 = 1$ wird bis zum Endzeitpunkt $T = 1.64$ gelöst mit der Schrittweite $\Delta t = 0.01$. Dadurch gibt es 64 Zeitschritte. Für alle Speeduptests mit $N_p < 64$ müssen also mehrere Pararealblöcke ausgeführt werden. Auf jedem Kern läuft nur ein Prozess, sodass kein *Hyperthreading* verwendet wird. Da jeder Knoten des Clusters 16 Kerne enthält, wurden bis zu 4 Knoten für die Speeduptests verwendet. Für die Kommunikation zwischen den Prozessen wird das *Message Passing Interface (MPI)* eingesetzt. Das *PFASST++*-Framework bietet Funktionen für das blockierende und nicht-blockierende Senden und Empfangen zwischen benachbarten *MPI-ranks*. Die Parareal-Implementierungen verwenden ausschließlich blockierende Kommunikation.

Um die theoretischen maximalen Speedups des klassischen und der hybriden Parareal/-SDC-Varianten berechnen zu können, wurde das in Abschnitt 2.1.4 definierte Laufzeitverhältnis α für die drei Vergröberungsstrategien gemessen. Das Laufzeitverhältnis für den klassischen Parareal wird mit α_c bezeichnet und wird ermittelt, indem die Laufzeit eines SDC-Laufs mit mehreren Iterationen sowohl mit der groben als auch mit der feinen Konfiguration für ein Zeitschritt mehrmals gemessen wird und der Mittelwert berechnet wird. Das Laufzeitverhältnis für die hybriden Parareal/SDC-Varianten, bezeichnet mit α_h wird im Gegensatz dazu über mehrere Laufzeitmessungen nur einer SDC-Iteration mit der feinen und der groben Konfiguration ermittelt. In Tabelle 4.3 sind die Laufzeitverhältnisse für die drei Vergröberungsstrategien aufgelistet. Um die theoretischen Speedups berechnen zu können, wird außerdem noch die Anzahl der benötigten Parareal-Iterationen K, bis die Konvergenzgenauigkeit erreicht wurde, gebraucht. Für die Speeduptests wurde die Toleranz für die Abbruchbedingung $\epsilon = 10^{-10}$ gewählt. Da K nur nach der Ausführung von Parareal bekannt ist, können die theoretischen Speedups auch nur nach der Ausführung, also „a posteriori“ berechnet werden. In den Plots von den gemessenen Speedups werden die theoretischen Speedups eingezeichnet um einen Vergleich zwischen der Theorie und der Praxis ziehen zu können. Im nächsten Abschnitt werden die Vergröberungsstrategien unter verschiedenen Gesichtspunkten analysiert.

Vergröberung	α_c	α_h
nur räumlich	0.53	0.55
nur zeitlich	0.61	0.46
kombiniert	0.33	0.26

Tabelle 4.3: Gemessene Laufzeitverhältnisse für die drei Vergröberungsstrategien.

Der theoretische Speedup von Parareal hängt, wie in Abschnitt 2.1.4 dargelegt wurde, von zwei Faktoren ab, nämlich dem Laufzeitverhältnis α und der Anzahl der Parareal-Iterationen K. K wiederum ist abhängig davon, wie grob der grobe Propagator im Vergleich zum feinen Propagator ist. Je gröber der grobe Propagator ist, also je kleiner α ist, desto mehr Iterationen werden benötigt, um die gewünschte Genauigkeit zu erreichen. Desweiteren hat die Anzahl der Prozesse N_p einen Einfluss auf K. Je mehr Prozesse eingesetzt werden, desto mehr muss iteriert werden, damit die gewünschte Genauigkeit auch im Zeitschritt des letzten Prozesses erreicht wird. Dieser Einfluss wird anhand von den Messungen der Iterationszahlen deutlich. Im Folgenden werden die verschiedenen Vergröberungsstrategien untereinander verglichen. Zu jeder Strategie werden in den jeweiligen Unterabschnitten die Speedupmessungen, die Iterationszahlen, die Fehler im letzten Zeitpunkt in jeder Iteration und die Laufzeitanalyse dargelegt. Abschließend werden die Ergebnisse zusammengefasst.

4.3.1 Nur räumliche Vergröberung

In Abbildung 4.4 wird die Speedupmessung für die rein räumlichen Vergröberung dargestellt. Neben den gemessenen Speedups der drei Parareal-Implementierungen werden auch die berechneten theoretischen Speedups angezeigt. Es ist deutlich sichtbar, dass die gemessenen Speedups mit den theoretischen Speedups ziemlich gut übereinstimmen. Bemerkenswert ist, dass die beiden hybriden Parareal/SDC-Varianten fast gleiche Speedups erzielen. Die Übereinstimmung der Speedups ist dadurch zu erklären, dass die Iterationszahlen der beiden hybriden Parareal/SDC-Varianten sich kaum unterscheiden. Beispielsweise benötigt der teil-hybride Parareal/SDC 14 Iterationen und der voll-hybride Parareal/SDC 12 Iterationen. Dadurch ergibt sich eine ähnliche Laufzeit und somit ähnliche Speedups.

In Abbildung 4.5 werden die Iterationszahlen von den drei Parareal-Implementierungen zu jedem Zeitschritt gezeigt. Die Abbildung enthält die Messungen mit 16, 32 und 64 Kerne. Es wird aus der Abbildung ersichtlich, dass in den späteren Zeitschritten immer mehr iteriert werden muss als in den ersten Zeitschritten. Dies ist dadurch zu erklären, dass die Näherungen über die Zeit immer ungenauer sind. Auch wird deutlich, dass beim Einsatz von mehr Kernen auch mehr iteriert werden muss, bis die gewünschte Genauigkeit erreicht ist. Der klassische Parareal benötigt nur eine Iteration, um die

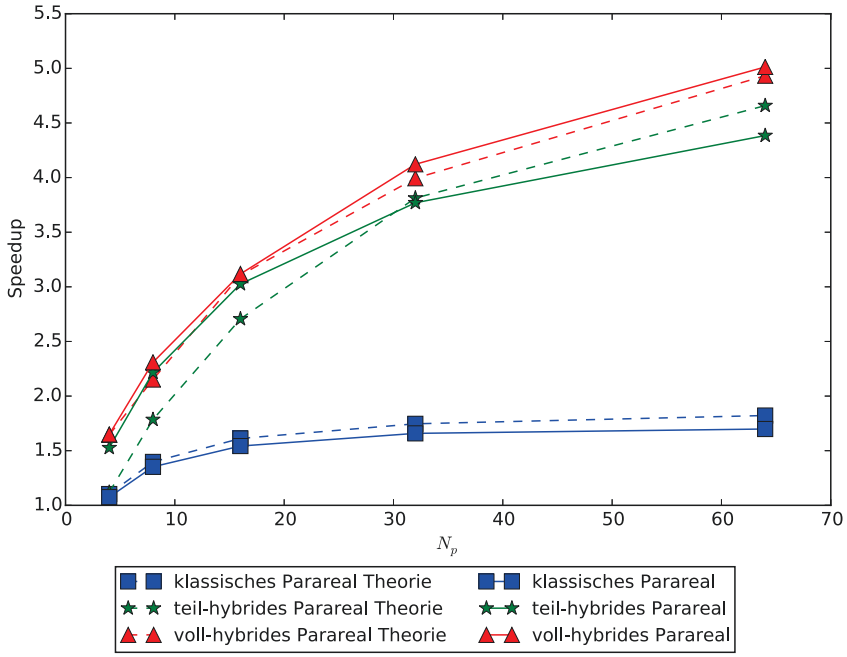

Abbildung 4.4: Speedup-Messung mit rein räumlicher Vergröberung.

gewünschte Genauigkeit zu erreichen. Da das grobe SDC die selbe Anzahl an Quadraturknoten verwendet wie das feine SDC, unterscheiden sich die Näherungen des groben SDC Propagators nur durch die geringere räumliche Auflösung. Die schnelle Konvergenz des klassischen Parareal ist ein Indiz dafür, dass sich die geringere räumliche Auflösung des groben SDC Propagators nicht sonderlich negativ auf die Genauigkeit der Näherung auswirkt.

Dies sieht man auch an den Fehlern in jeder Iteration. In Abbildung 4.6 ist der Fehler der Näherung im Endzeitpunkt in jeder Iteration für die beiden hybriden Parareal/SDC-Varianten dargestellt. Dabei variiert N_p wie in Abbildung 4.5 von 16 bis 64. Das in Abschnitt 4.2 beobachtete Konvergenzproblem des voll-hybriden Parareal/SDC tritt hier nicht auf, da nur räumliche Vergröberung eingesetzt wird und somit die Propagatoren die gleiche Fehlerordnung besitzen. Es wird deutlich, dass der Fehler des teil-hybriden Parareal/SDC genauso schnell abfällt wie der Fehler des voll-hybriden Parareal/SDC. Außerdem kann beobachtet werden, dass der Fehler bei beiden Parareal-Varianten mit steigendem N_p langsamer fällt, sodass mehr iteriert werden muss.

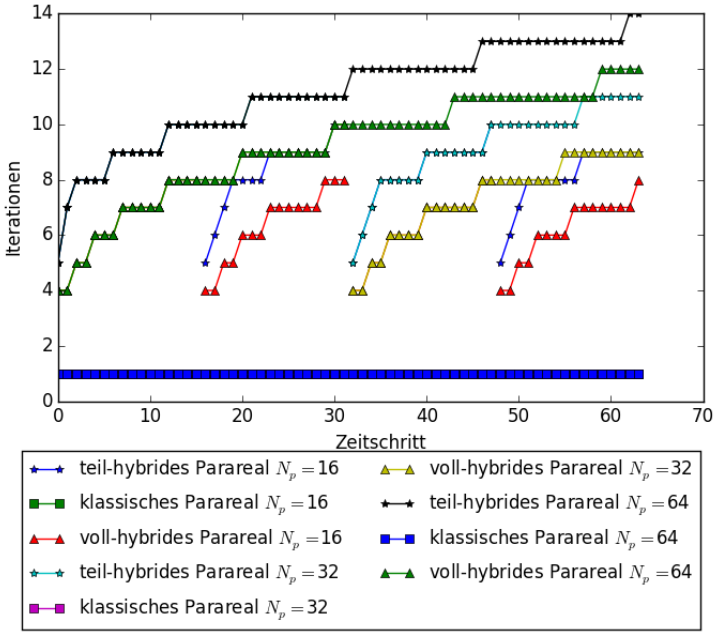

Abbildung 4.5: Iterationszahlen der drei Parareal-Implementierungen für die rein räumliche Vergröberung.

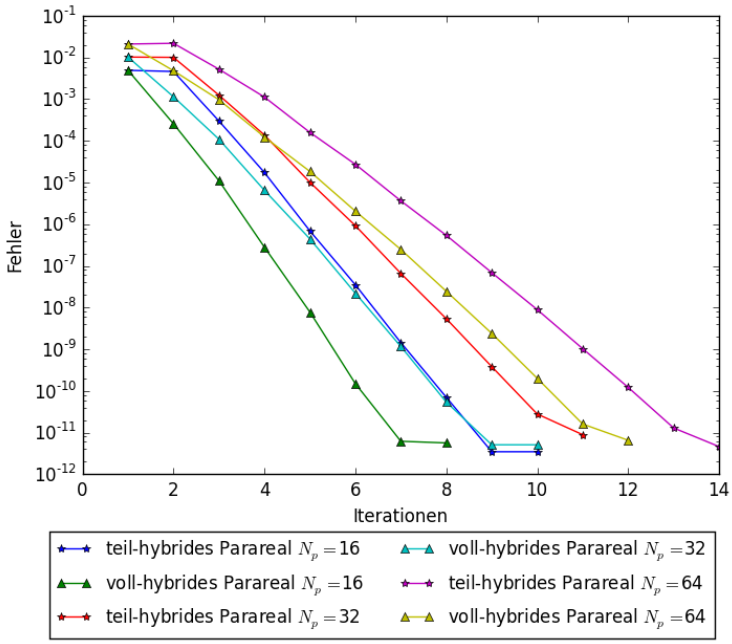

Abbildung 4.6: Fehler der Näherung im letzten Zeitschritt in allen Iterationen von den hybriden Parareal/SDC Varianten für die rein räumliche Vergröberung.

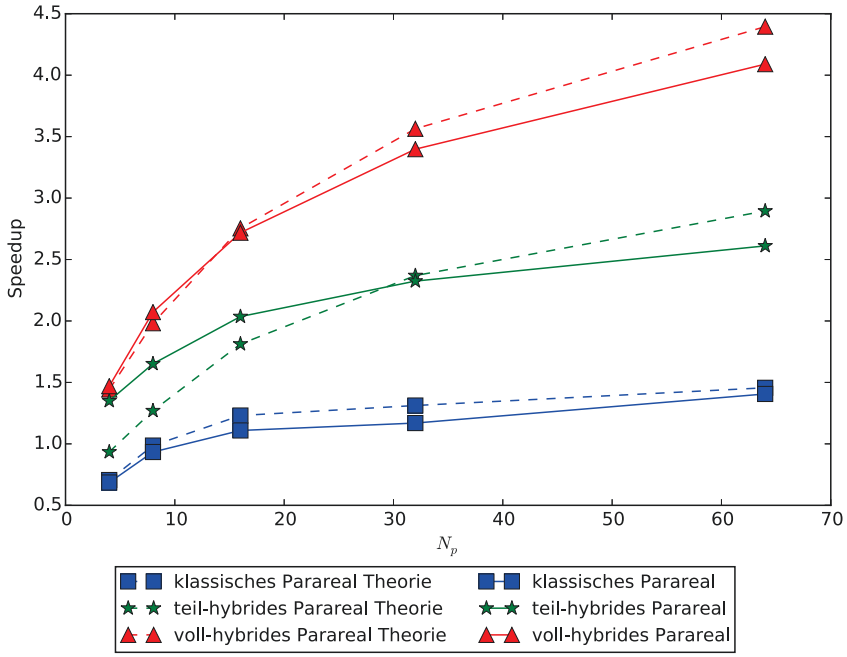

Abbildung 4.7: Speedup-Messung mit rein zeitlicher Vergröberung.

4.3.2 Nur zeitliche Vergröberung

Das hybride Laufzeitverhältnis α_h für die rein zeitliche Vergröberung ist etwas geringer als für die rein räumliche Vergröberung, wobei dies beim klassischen Laufzeitverhältnis α_c umgekehrt ist. Ein weiterer wesentlicher Unterschied zu der rein räumlichen Vergröberung ist, dass die Fehlerordnung des groben SDC aufgrund der geringeren Anzahl der Quadraturknoten ($\tilde{J} < \hat{J}$) kleiner ist als die Fehlerordnung des feinen SDC. Dadurch dass der grobe Propagator wegen der geringeren Fehlerordnung ungenauere Näherungen berechnet, ist eine höhere Iterationszahl K für die Konvergenz vonnöten. In Abbildung 4.7 werden die Speedupmessungen der drei Parareal-Implementierungen für die rein zeitliche Vergröberung dargestellt. Es zeigt sich, dass die Speedups der beiden hybriden Parareal/SDC-Varianten im Gegensatz zu der rein räumlichen Vergröberung einen starken Unterschied aufweisen. Der Speedup des voll-hybriden Parareal/SDC ist annähernd 60% höher als der Speedup des teil-hybriden Parareal/SDC. Außerdem gibt es eine Diskrepanz zwischen den gemessenen und den theoretischen Speedups, die für größeres N_p deutlicher wird. Die Erklärung dafür ist, dass die Iterationszahl deutlich höher ist als bei der rein räumlichen Vergröberung und dadurch mehr Kommunikation zwischen den Kernen vonnöten ist, was sich wiederum negativ auf den Speedup auswirkt. Die Kommunikation und Interpolation bzw. Restriktion wurden bei der Herleitung des theoretischen Speedups nicht berücksichtigt, sodass eine Abweichung von den gemessenen Speedups nicht verwunderlich ist. Beim teil-hybriden Parareal/SDC fällt auf, dass die Speedups

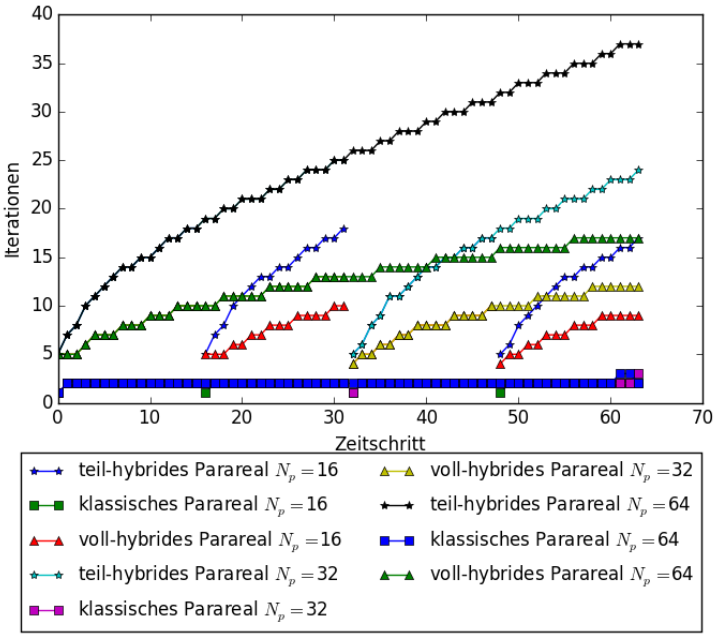

Abbildung 4.8: Iterationszahlen der drei Parareal-Implementierungen für die rein zeitliche Vergröberung.

für $N_p < 32$ höher sind als die theoretischen Speedups. Die Ursache dafür liegt in der Implementierung des teil-hybriden Parareal/SDC, da im Gegensatz zu der Theorie der erste und der letzte Prozess nur den feinen Propagator ausführen, da sie keine Korrektur der Anfangswerte an den nächsten Prozess weiterreichen müssen. Durch die Einsparung der Ausführung des groben Propagators ist die tatsächliche Laufzeit geringer als die theoretische, worin alle Prozesse beide Propagatoren ausführen.

In Abbildung 4.8 werden die Iterationszahlen für die rein zeitliche Vergröberung dargestellt. Wie erwartet sind die Iterationszahlen höher als bei der rein räumlichen Vergröberung. Außerdem wird deutlich, dass die Iterationszahlen des teil-hybriden Parareal/SDC höher sind als die des voll-hybriden Parareal/SDC, wodurch der geringere Speedup des teil-hybriden Parareal/SDC verursacht wird. Wie bei der rein räumlichen Vergröberung steigt die Iterationszahl mit N_p.

In Abbildung 4.9 sieht man, dass der Fehler des voll-hybriden Parareal/SDC, wie schon in Abschnitt 4.2 beim Konvergenztest mit kombinierter Vergröberung gezeigt wurde, gegen den Fehler des groben SDC konvergiert. Das Konvergenzproblem tritt immer dann auf, wenn der Fehler des groben Propagators vom Fehler des feinen Propagators abweicht, was in der Regel der Fall ist. Die Abbildung demonstriert zudem die Unterschiede der beiden hybriden Parareal/SDC-Varianten hinsichtlich der Iterationszahlen. Der Fehler des voll-hybriden Parareal/SDC fällt schneller als der Fehler des teil-hybriden Parareal/SDC,

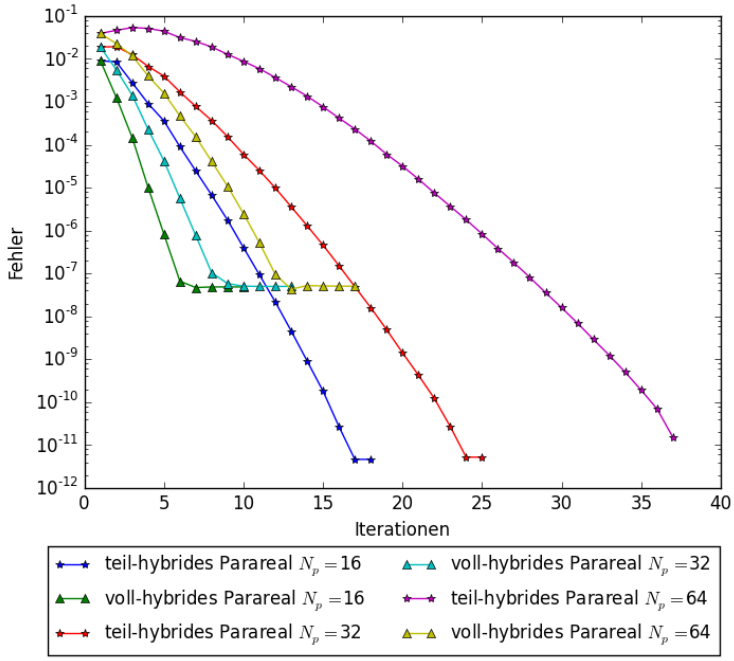

Abbildung 4.9: Fehler der Näherung im letzten Zeitschritt in allen Iterationen von den beiden hybriden Parareal/SDC-Varianten für die rein zeitliche Vergröberung.

was durch die steileren Fehlerkurven gut ersichtlich ist. Der teil-hybride Parareal/SDC konvergiert zwar wie aus der Abbildung ersichtlich wird gegen den Fehler des feinen SDC, aber die Iterationszahl steigt mit ansteigendem N_p immens an. Dadurch wird der maximal erreichbare Speedup stark limitiert.

4.3.3 Kombinierte Vergröberung

Die kombinierte Vergröberung ist eine Kombination der beiden einzelnen Vergröberungsstrategien und damit eine stärkere Vergröberung. Dadurch können geringere Laufzeitverhältnisse α_c und α_h erzielt werden, wie der Tabelle 4.3 entnommen werden kann. Die Laufzeitverhältnisse sind annähernd die Hälfte der Laufzeitverhältnisse der rein zeitlichen Vergröberung, was durch die Halbierung der räumlichen Auflösung bewirkt wird. Es werden also höhere Speedups erwartet als bei der rein zeitlichen Vergröberung, vorausgesetzt, dass die Iterationszahlen gleich bleiben. In Abbildung 4.10 sind die gemessenen Speedups der drei Parareal-Implementierungen und die theoretischen Speedups für die kombinierte Vergröberung dargestellt. Sowohl die gemessenen Speedups als auch die Diskrepanz zwischen den theoretischen und gemessenen Speedups sind hierbei höher als in der Abbildung 4.7, worin die Speedups der Parareal-Implementierungen für die

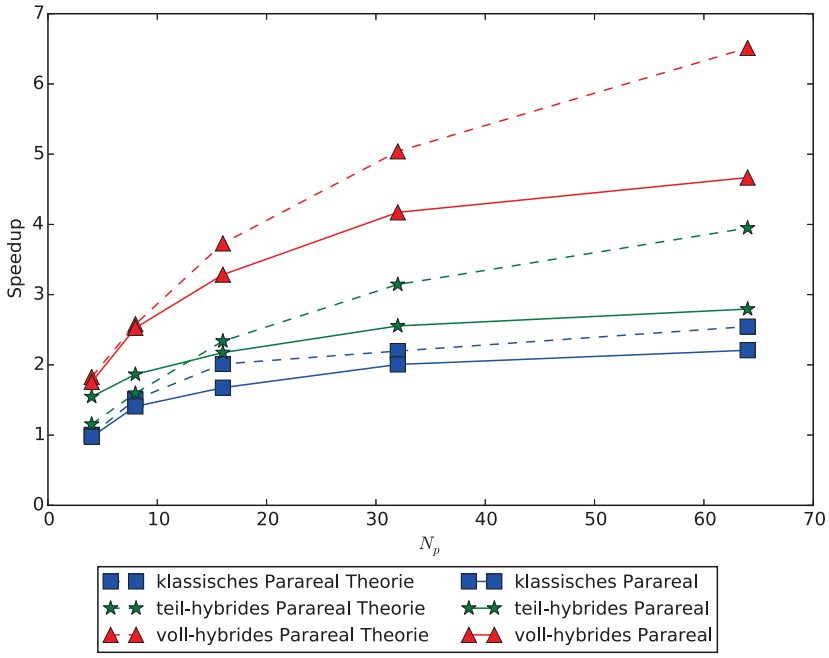

Abbildung 4.10: Speedup-Messung für die kombinierte Vergröberung.

rein zeitliche Vergröberung dargestellt sind. Diese Diskrepanz wird durch die höheren Laufzeiten der Interpolation verursacht, da nun zusätzlich zu der zeitlichen Interpolation auch räumlich interpoliert wird. Die Iterationszahlen bestimmen zudem die Gesamtlaufzeit der Interpolation, da in jeder Iteration die Näherungen des groben SDC Propagators interpoliert werden müssen.

Die Iterationszahlen sind in Abbildung 4.11 dargestellt. Diese sind mit den Iterationszahlen der Parareal-Implementierungen mit rein zeitlicher Vergröberung identisch. Daraus lässt sich schließen, dass die räumliche Auflösung des groben SDC Propagators nicht in ungenaueren Näherungen resultiert, da sonst mehr iteriert werden müsste, um die zusätzliche Ungenauigkeit durch die räumliche Vergröberung zu amortisieren. Dadurch dass die Laufzeitverhältnisse der kombinierten Vergröberung wie angesprochen geringer sind als die Laufzeitverhältnisse der rein zeitlichen Vergröberung bei gleichbleibender Iterationszahl können theoretisch höhere Speedups erzielt werden. Diese werden aber aus bereits erwähnten Gründen in der Praxis nicht erreicht.

Die Fehler der letzten Näherungen der beiden hybriden Parareal/SDC-Varianten verhalten sich im Falle der kombinierten Vergröberung wie die Fehler bei der rein zeitlichen Vergröberung, wie der Abbildung 4.12 entnommen werden kann. Das Konvergenzproblem des voll-hybriden Parareal/SDC taucht hier ebenfalls auf, da der grobe SDC Propagator wie bei der rein zeitlichen Vergröberung eine niedrigere Fehlerordnung besitzt. Die zusätzliche räumliche Vergröberung hat also keine Auswirkung auf den Fehler.

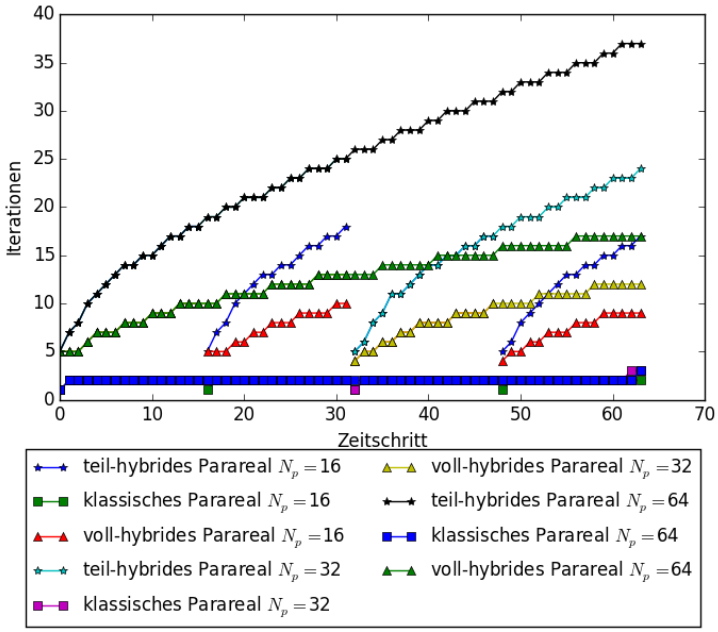

Abbildung 4.11: Iterationszahlen der drei Parareal-Implementierungen für die kombinierte Vergröberung.

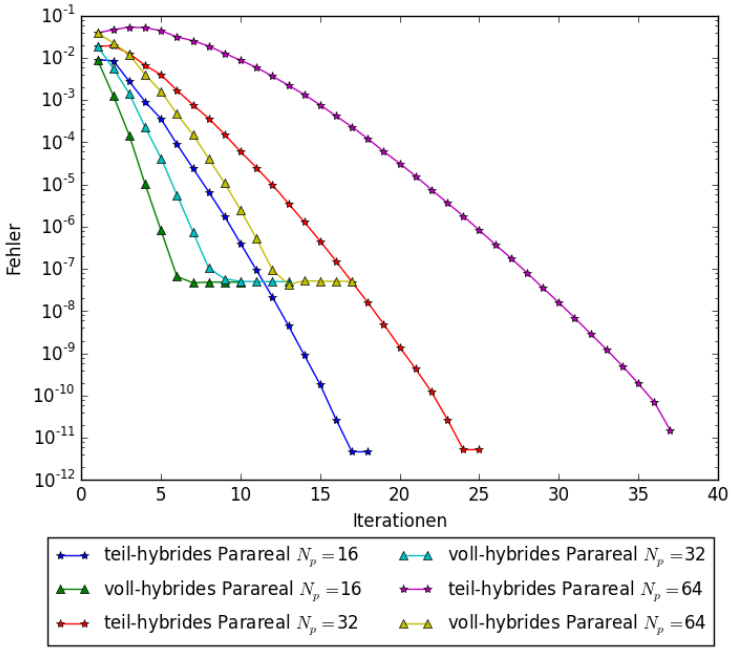

Abbildung 4.12: Fehler der Näherung im letzten Zeitschritt in allen Iterationen von den beiden hybriden Parareal/SDC-Varianten für die kombinierte Vergröberung.

Parareal-Variante	**Vergröberung**	K_{64}	S_{64}
klassisches Parareal	nur räumlich	1	1.70
	nur zeitlich	3	1.41
	kombiniert	3	2.21
teil-hybrides Parareal/SDC	nur räumlich	14	4.39
	nur zeitlich	37	2.61
	kombiniert	37	2.8
voll-hybrides Parareal/SDC	nur räumlich	12	5.01
	nur zeitlich	17	4.09
	kombiniert	17	4.67

Tabelle 4.4: Maximale Speedups S_{64} und maximale Iterationszahlen K_{64} bei 64 Kernen für die drei Parareal-Implementierungen und für die drei Vergröberungsstrategien.

4.3.4 Zusammenfassung

Der Vergleich der drei Vergröberungsstrategien hinsichtlich der Auswirkung auf den erzielten Speedup verdeutlicht die Unterschiede zwischen den Parareal-Implementierungen. Die Ergebnisse bestätigen die Annahme, dass die hybriden Parareal/SDC-Varianten bessere Speedups erzielen als die klassische Parareal Version. In Tabelle 4.4 sind die in den numerischen Tests unter Verwendung von 64 Kerne erreichten maximalen Speedups S_{64} für die drei Parareal-Implementierungen aufgelistet. Der Index beschreibt dabei die Anzahl der Kerne. Je Parareal-Implementierung werden die Ergebnisse für die drei Vergröberungsstrategien angegeben. Neben den Speedups sind außerdem die für die Konvergenz benötigte Anzahl an Iterationen K_{64} eingetragen, da die Speedups stark davon abhängen. Es ist deutlich erkennbar, dass die voll hybride Parareal/SDC-Variante die besten Speedups erzielt. Dahinter liegt die teil-hybride Parareal/SDC-Variante und die klassische Parareal-Implementierung erreicht deutlich niedrigere Speedups. Die unterschiedlichen Speedups der beiden hybriden Parareal/SDC-Varianten werden hauptsächlich durch die zum Teil großen Unterschiede in den Iterationszahlen verursacht. Die Unterschiede fallen für die verschiedenen Vergröberungsstrategien unterschiedlich hoch aus. Bei der rein räumlichen Vergröberung weichen die Iterationszahlen der beiden hybriden/SDC Parareal-Varianten, wie schon angesprochen wurde, kaum voneinander ab. Durch die zwei zusätzlichen Iterationen, die teil-hybride Parareal/SDC Implementierung für die Konvergenz braucht, ergibt sich ein minimal geringerer Speedup von 0.7. Für die beiden anderen Vergröberungsstrategien sind die Abweichungen der Iterationszahlen der beiden hybriden Parareal/SDC-Varianten jedoch immens. Die teil-hybride Parareal/SDC-Variante benötigt mehr als doppelt so viele Iterationen als die voll-hybride/SDC-Variante, was sich auch in deutlich niedrigeren Speedups ausdrückt. Es muss jedoch berücksichtigt werden, dass die voll-hybride Parareal/SDC-Variante aufgrund des angesprochenen Konvergenzproblems im Gegensatz zu der teil-hybriden Va-

riante nicht den Fehler des feinen SDC Propagators erreicht. Aufgrund dessen kann ein direkter Vergleich der Iterationszahlen irreführen. Es kann jedoch angenommen werden, dass die voll-hybride Parareal/SDC-Variante weniger Iterationen für die Konvergenz benötigt, was durch die schnellere Abnahme des Fehler im Vergleich zur teilhybriden Parareal/SDC-Variante, wie in den Abbildungen 4.9 und 4.12 ersichtlich wird, bewirkt wird. Der Unterschied zwischen der rein zeitlichen und kombinierten Vergröberung fällt kaum ins Gewicht, wie in Abschnitt 4.3.3 besprochen wurde. Ein wesentlicher Unterschied zwischen der klassischen und den hybriden Parareal/SDC Implementierungen wird beim Vergleich der maximalen Speedups ersichtlich. Die klassische Parareal-Implementierung erreicht den maximalen Speedup bei der kombinierten Vergröberung während die beiden hybriden Parareal/SDC Implementierungen den maximalen Speedup bei der rein räumlichen Vergröberung erzielen. Die klassische Parareal-Implementierung bestätigt die intuitive Annahme, dass eine stärkere Vergröberung auch einen höheren Speedup zur Folge hat. In der Hinsicht verhalten sich die beiden hybriden Parareal/SDC-Varianten kontraintuitiv. Dieses unerwartete Verhalten wird durch den höheren Berechnungsaufwand aufgrund der zusätzlichen zeitlichen Interpolation verursacht. Es ist jedoch nicht auszuschließen, dass die zusätzliche zeitliche Vergröberung bei anderen Anfangswertproblemen auch zu höheren Speedups der hybriden Parareal/SDC-Varianten führt. Bei der Verwendung der zeitlichen und kombinierten Vergröberungsstrategien in Verbindung mit den hybriden Parareal/SDC-Varianten muss also der zusätzliche Aufwand durch die Interpolation berücksichtigt werden. Die Interpolation ist auch zusammen mit dem Kommunikation der Grund für die Abweichung der gemessenen Speedups von den theoretisch maximal erreichbaren Speedups. Die Abweichungen sind umso höher je mehr Iterationen für die Konvergenz benötigt werden, da die Ausführungszeiten der Interpolation und Kommunikation mit der Iterationszahl skalieren. Die theoretische Steigerung des Speedups des klassischen Parareal um das M-fache durch das hybride Parareal/SDC, wobei M die Anzahl der benötigten Iterationen für die Konvergenz des seriellen feinen SDC-Verfahrens ist, konnte nicht beobachtet werden, da die Iterationszahlen dafür gleich bleiben müssten. Die Iterationszahlen der hybriden Parareal/SDC-Varianten sind jedoch ein Vielfaches der Iterationszahlen des klassischen Parareal.

5 Fazit und Ausblick

In diesem Kapitel werden die Ergebnisse der vorliegenden Arbeit zusammengefasst und hinsichtlich ihrer Bedeutung für die aktuelle Forschung im Bereich der zeitparallelen Algorithmen bewertet. Die hybriden Parareal/SDC Implementierungen erreichen für das Advektions-Diffusions-Problem in den numerischen Tests insgesamt höhere Speedups als die Implementierung des klassischen Parareal mit voll auskonvergierendem SDC als Propagator, wodurch die in der Theorie besagte Erhöhung des Speedups bestätigt werden konnte. Die Auswertung der numerischen Ergebnisse im Rahmen dieser Arbeit förderten jedoch unterschiedliche Konvergenzverhalten der beiden hybriden Parareal/SDC-Varianten zu Tage. Der Fehler der voll-hybriden Parareal/SDC-Variante konvergiert gegen den Fehler des groben Propagators anstatt gegen den des feinen Propagators, wie von Parareal eigentlich erwartet wird. Dieses Konvergenzproblem wird durch die Interpolation der Näherungen des groben SDC Propagators auf die Näherungen des feinen SDC Propagators an allen Knoten ohne Berücksichtigung der geringeren Genauigkeit des groben SDC Propagators verursacht. Die Genauigkeit des feinen SDC Propagators geht durch die Interpolation verloren. Das Prinzip der Verbindung verschiedener Diskretisierungen eines Problems für die Verringerung des numerischen Rechenaufwandes wird von den Mehrgitterverfahren befolgt. Die Mehrgitterverfahren setzen eine Grobgitterkorrektur ein, um die Genauigkeit der Näherungen auf der feinen Diskretisierung zu erhalten.

Dieser Ansatz aus der Theorie der Mehrgitterverfahren wurde verfolgt, um das Konvergenzproblem des voll-hybriden Parareal/SDC zu überwinden. Der *PFASST*-Algorithmus (*Parallel Full Approximation Scheme in Space and Time*) [11] ist die Weiterentwicklung des voll-hybriden Parareal/SDC mit dem Zusatz der *FAS-Korrektur (Full Approximation Scheme)* [18], die von den nichtlinearen Mehrgitterverfahren für den Transfer der Näherungen zwischen den unterschiedlichen Diskretisierungen verwendet wird. Im nächsten Abschnitt wird die FAS-Korrektur wie in [18] rekapituliert und dessen Einsatz im PFASST-Algorithmus beschrieben.

5.1 Bedeutung der FAS-Korrektur

Die FAS-Korrektur ist Teil des Mehrgitterverfahrens für nichtlineare Gleichungen der Form:

$$A(u) = f, \qquad u, f \in \mathbb{R}^n. \tag{5.1}$$

Eine Näherung der exakten Lösung u wird mit v bezeichnet und der Fehler kann dann über die Fehlergleichung $e = u - v$ angegeben werden. Das Residuum r wird ausgedrückt durch

$$r = A(u) - A(v) \Leftrightarrow r = A(e + v) - A(v). \tag{5.2}$$

Die auf dem feinen Gitter diskretisierte Gleichung (5.1) wird durch $A^h(u^h) = f^h$ ausgedrückt. Die grobe Diskretisierung wird dementsprechend durch $A^H(u^H) = f^H$ beschrieben. Die grobe Version der Residuumsgleichung (5.2) ist gegeben durch:

$$A^H(v^H + e^H) - A^H(v^H) = r^H \tag{5.3}$$

Das Residuum des groben Gitters wird folgendermaßen als Restriktion des Residuums des feinen Gitters definiert, wobei I_h^H den Transfer von feinen zum groben Gitter beschreibt:

$$r^H = I_h^H r^h = I_h^H (f^h - A^h(v^h)). \tag{5.4}$$

Dementsprechend wird die Näherung auf dem feinen Gitter mit dem selben Transferoperator restringiert, sodass für die grobe Näherung gilt: $v^H = I_h^H v^h$. Eingesetzt in die Residuumsgleichung des groben Gitters ergibt dies:

$$A^H(\underbrace{I_h^H v^h + e^H}_{u^H}) = \underbrace{A^H(I_h^H v^h) + I_h^H \left(f^h - A^h(v^h)\right)}_{f^H}. \tag{5.5}$$

Die rechte Seite von dieser Gleichung ist bekannt und wenn nun die Lösung u^H gefunden wird, kann der Fehler des groben Gitters bestimmt werden über $e^H = u^H - I_h^H v^h$. Der Fehler wird dann interpoliert, um die Näherung des feinen Gitters v^h zu korrigieren:

$$v^h \leftarrow v^h + I_H^h e^H. \tag{5.6}$$

Dieses Verfahren heißt *Full Approximation Scheme*, da die nichtlineare Gleichung des groben Gitters (5.5) nach der vollen Näherung u^H gelöst wird anstatt nach dem Fehler e^H, wie es bei den linearen Mehrgitterverfahren der Fall ist. Eine andere Schreibweise der Gleichung auf dem groben Gitter ist

$$A^H(u^H) = f^H + \tau_h^H \qquad \text{wobei} \qquad \tau_h^H = A^H(I_h^H v^h) - I_h^H A^h(v^h) \tag{5.7}$$

die so genannte *Tau-Korrektur* ist. Durch diese Korrektur ist es möglich, die Genauigkeit der Näherung auf dem groben Gitter auf ein mit der Genauigkeit der Näherung auf dem feinen Gitter vergleichbares Niveau zu heben.

Der PFASST-Algorithmus setzt die FAS-Korrektur ein, um die Genauigkeit der Näherungen des groben SDC Propagators zu erhöhen um so das Konvergenzproblem des voll-hybriden Parareal/SDC zu beheben. Nach jedem Aufruf der *Transfer*-Funktion `restrict` wird die Tau-Korrektur berechnet und auf die rechte Seite addiert. Dadurch wird in der nächsten Iteration des groben SDC Propagators die Korrektur automatisch einbezogen. Diese Berechnung behebt das Konvergenzproblem der voll-hybriden Parareal/SDC-Variante und stellt somit eine qualitative Weiterentwicklung dessen dar. Die ursprünglich veröffentlichte Version von PFASST [11] verwendet zwei Level bzw. Diskretisierungen wie in Parareal. Wenig später wurde PFASST auch für mehrere Level definiert [12], wobei die verschiedenen Level bzw. Gitter in einem V-Zyklus durchlaufen werden wie in MLSDC (Multi Level SDC) [33]. Eine alternative Betrachtungsweise ist, dass PFASST eine zeitparallele Version von MLSDC ist, die iterativ die parallel berechneten Näherungen seriell korrigiert.

5.2 Zusammenfassung und Ausblick

Die vorliegende Arbeit legt die Entwicklung der beiden hybriden Parareal/SDC-Algorithmen dar und zeigt numerische Resultate der Implementierung dieser Algorithmen im PFASST++-Framework. Durch die Implementierung der drei in dieser Arbeit geschilderten Parareal-Varianten im selben Framework wurde ein systematischer Vergleich untereinander möglich. Dadurch konnten eindeutige Resultate, wie die höheren Speedups der hybriden Parareal/SDC-Varianten und das Konvergenzproblem des voll-hybriden Parareal/SDC, erzielt werden. Die Arbeit stellt somit ein Verbindung vom ursprünglichen Parareal zu den aktuellen darauf aufbauenden Entwicklungen her und demonstriert deren Verbesserungen bzw. Probleme. Die in Kapitel 4 dargelegte Konvergenzanalyse der drei Parareal-Implementierungen brachte als Resultat das in dieser Arbeit als Konvergenzproblem bezeichnete fehlerhafte Konvergenzverhalten der voll-hybriden Parareal/SDC-Variante hervor. Um dieses Problem zu lösen, wurde der hybride Parareal/SDC-Algorithmus mit Einbeziehung der aus den nichtlinearen Mehrgitterverfahren entliehenen FAS-Korrektur konsequent weiterentwickelt. Diese als PFASST-Algorithmus [11] bekannte Weiterentwicklung des hybriden Parareal/SDC-Algorithmus wird aktuell unter anderem am Forschungszentrum Jülich erforscht und für den Einsatz in verschiedenen Bereichen erprobt [26,32,34]. Parareal ist trotz der eher mageren parallelen Effizienz noch stärker verbreitet in der Anwendung als PFASST [3,4,16,29]. Dies ist jedoch zum Teil auf die einfachere Implementierbarkeit von Parareal zurückzuführen, da Parareal bereits existierende Implementierungen von Einschrittverfahren/Propagatoren als Blackbox-Elemente verwenden kann. Die Implementierung von PFASST ist hingegen wie bei den hybriden Parareal/SDC-Varianten durch die Verwebung mit SDC eher aufwändig. PFASST hat alles in Allem bessere Zukunftsaussichten, da es die bessere Performance bietet und durch die Verknüpfung mit Mehrgitterverfahren im Raum [26] auch weitere Effizienzsteigerungen erlaubt.

Literaturverzeichnis

[1] *List of publications related to parallel-in-time integration.* http://www.parallelintime.org/references/index.html. Version: 2015

[2] *List of the top 500 supercomputers.* http://www.top500.org/statistics/. Version: 2015

[3] Ariel, G. ; Kim, S. J. ; Tsai, R.: *Parareal methods for highly oscillatory ordinary differential equations.* arXiv:1503.02094 [math.NA]. http://arxiv.org/abs/1503.02094v1. Version: 2015

[4] Arteaga, A. ; Ruprecht, D. ; Krause, R.: A stencil-based implementation of Parareal in the C++ domain specific embedded language STELLA. In: *Applied Mathematics and Computation* 267 (2015), 727–741. http://dx.doi.org/10.1016/j.amc.2014.12.055

[5] Baudron, A. ; Lautard, J. ; Maday, Y. ; Mula, O.: The parareal in time algorithm applied to the kinetic neutron diffusion equation. In: *Domain Decomposition Methods in Science and Engineering XXI*, Springer International Publishing, 2014 (Lecture Notes in Computational Science and Engineering), 437–445

[6] Bellen, A. ; Zennaro, M.: Parallel algorithms for initial-value problems for difference and differential equations. In: *Journal of Computational and Applied Mathematics* 25 (1989), Nr. 3, 341–350. http://dx.doi.org/10.1016/0377-0427(89)90037-X

[7] Böhmer, K. ; Hemker, P.W. ; Stetter, H.J.: The Defect Correction Approach. Version: 1984. http://dx.doi.org/10.1007/978-3-7091-7023-6_1. In: Böhmer, Klaus (Hrsg.) ; Stetter, HansJ. (Hrsg.): *Defect Correction Methods* Bd. 5. Springer Vienna, 1984. – ISBN 978–3–211–81832–9, 1-32

[8] Briggs, W. L. ; Henson, Van E. ; McCormick, S. F.: *A Multigrid Tutorial (2Nd Ed.).* Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 2000. – ISBN 0–89871–462–1

[9] Chartier, P. ; Philippe, B.: A parallel shooting technique for solving dissipative

ODE's. In: *Computing* 51 (1993), Nr. 3-4, 209–236. http://dx.doi.org/10.1007/BF02238534

[10] Dutt, A. ; Greengard, L. ; Rokhlin, V.: Spectral Deferred Correction Methods for Ordinary Differential Equations. In: *BIT Numerical Mathematics* 40 (2000), Nr. 2, 241-266. http://dx.doi.org/10.1023/A:1022338906936. – DOI 10.1023/A:1022338906936. – ISSN 0006–3835

[11] Emmett, M. ; Minion, M. L.: Toward an Efficient Parallel in Time Method for Partial Differential Equations. In: *Communications in Applied Mathematics and Computational Science* 7 (2012), 105–132. http://dx.doi.org/10.2140/camcos.2012.7.105

[12] Emmett, M. ; Minion, M. L.: Efficient implementation of a multi-level parallel in time algorithm. In: *Domain Decomposition Methods in Science and Engineering XXI* Bd. 98, Springer International Publishing, 2014 (Lecture Notes in Computational Science and Engineering), 359–366

[13] Gander, M. J.: 50 years of Time Parallel Time Integration. Version: 2015. http://www.unige.ch/%7Egander/Preprints/50YearsTimeParallel.pdf. In: *Multiple Shooting and Time Domain Decomposition*. Springer, 2015

[14] Gander, M. J. ; Hairer, E.: Nonlinear Convergence Analysis for the Parareal Algorithm. In: Langer, U. (Hrsg.) ; Widlund, O. (Hrsg.) ; Keyes, D. (Hrsg.): *Domain Decomposition Methods in Science and Engineering* Bd. 60, Springer, 2008 (Lecture Notes in Computational Science and Engineering), 45–56

[15] Greengard, L.: Spectral Integration and Two-Point Boundary Value Problems. In: *SIAM Journal on Numerical Analysis* 28 (1991), Nr. 4, 1071-1080. http://dx.doi.org/10.1137/0728057. – DOI 10.1137/0728057

[16] Gurrala, G. ; Dimitrovski, A. ; Sreekanth, P. ; Simunovic, S. ; Starke, M.: Parareal in Time for Dynamic Simulations of Power Systems. In: *Proceedings of the International Conference on Power Systems Transients (IPST2015) in Cavtat, Croatia June 15-18, 2015*, 2015

[17] Hackbusch, W.: Parabolic multi-grid methods. In: *Computing Methods in Applied Sciences and Engineering, VI* (1984), 189–197. http://dl.acm.org/citation.cfm?id=4673.4714

[18] Henson, Van E.: Multigrid Methods For Nonlinear Problems: An Overview. In: *Proceedings of the SPIE*, 2003, S. 36–48

[19] Klatt, Torbjörn ; Emmett, Matthew ; Ruprecht, Daniel ; Speck, Robert ;

Terzi, Selman: PFASST++: MPI Bugfix (v0.5.0). (2015), 06. http://dx.doi.org/10.6084/m9.figshare.1431794

[20] Krause, R. ; Ruprecht, D.: Hybrid Space-Time Parallel Solution of Burgers' Equation. In: *Domain Decomposition Methods in Science and Engineering XXI* Bd. 98, Springer International Publishing, 2014 (Lecture Notes in Computational Science and Engineering), 647–655

[21] Kreienbuehl, A. ; Naegel, A. ; Ruprecht, D. ; Speck, R. ; Wittum, G. ; Krause, R.: Numerical simulation of skin transport using Parareal. In: *Computing and Visualization in Science* (2015). http://dx.doi.org/10.1007/s00791-015-0246-y

[22] Lions, J.-L. ; Maday, Y. ; Turinici, G.d: A parareal̈n time discretization of PDE's. In: *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics* 332 (2001), 661-668. http://dx.doi.org/10.1016/S0764-4442(00)01793-6

[23] Loderer, T. ; Heuveline, V. ; Lohner, R.: The parareal algorithm as a new approach for numerical integration of ODEs in real-time simulations in automotive industry. In: *PAMM* 14 (2014), Nr. 1, 1027–1030. http://dx.doi.org/10.1002/pamm.201410489. – ISSN 1617–7061

[24] Minion, M. L.: Semi-implicit spectral deferred correction methods for ordinary differential equations. In: *Commun. Math. Sci.* 1 (2003), 09, Nr. 3, 471–500. http://projecteuclid.org/euclid.cms/1250880097

[25] Minion, M. L.: A Hybrid Parareal Spectral Deferred Corrections Method. In: *Communications in Applied Mathematics and Computational Science* 5 (2010), Nr. 2, 265–301. http://dx.doi.org/10.2140/camcos.2010.5.265

[26] Minion, M. L. ; Speck, R. ; Bolten, M. ; Emmett, M. ; Ruprecht, D.: Interweaving PFASST and parallel multigrid. In: *SIAM Journal on Scientific Computing* (2015). http://arxiv.org/abs/1407.6486

[27] Nievergelt, J.: Parallel methods for integrating ordinary differential equations. In: *Commun. ACM* 7 (1964), Nr. 12, 731–733. http://dx.doi.org/10.1145/355588.365137

[28] Ruprecht, D.: Convergence of Parareal with spatial coarsening. In: *PAMM* 14 (2014), Nr. 1, 1031–1034. http://dx.doi.org/10.1002/pamm.201410490. – ISSN 1617–7061

[29] Ruprecht, D. ; Speck, R. ; Krause, R.: Parareal for diffusion problems with space- and time-dependent coefficients. In: *Domain Decomposition Methods in*

Science and Engineering XXII Bd. 104, Springer International Publishing Switzerland, 2015 (Lecture Notes in Computational Science and Engineering), 3–10

[30] Schiesser, W. E. ; Griffiths, G. W.: *A Compendium of Partial Differential Equation Models: Method of Lines Analysis with Matlab*. 1. New York, NY, USA : Cambridge University Press, 2009. – ISBN 0521519861, 9780521519861

[31] Smith, B. ; Bjorstad, P. ; Gropp, W.: *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge University press, 2004

[32] Speck, R. ; Ruprecht, D. ; Emmett, M. ; Bolten, M. ; Krause, R.: A space-time parallel solver for the three-dimensional heat equation. In: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)* Bd. 25, IOS Press, 2014 (Advances in Parallel Computing), 263–272

[33] Speck, R. ; Ruprecht, D. ; Emmett, M. ; Minion, M. L. ; Bolten, M. ; Krause, R.: A multi-level spectral deferred correction method. In: *BIT Numerical Mathematics* 55 (2015), 843–867. http://dx.doi.org/10.1007/s10543-014-0517-x

[34] Speck, R. ; Ruprecht, D. ; Krause, R. ; Emmett, M. ; Minion, M. L. ; Winkel, M. ; Gibbon, P.: Integrating an N-body problem with SDC and PFASST. In: *Domain Decomposition Methods in Science and Engineering XXI* Bd. 98, Springer International Publishing, 2014 (Lecture Notes in Computational Science and Engineering), 637–645

[35] Speck, R. ; Ruprecht, D. ; Minion, M. ; Emmett, M. ; Krause, R.: Inexact spectral deferred corrections. In: *Domain Decomposition Methods in Science and Engineering XXII* Bd. 104, Springer International Publishing Switzerland, 2015 (Lecture Notes in Computational Science and Engineering), 127–133

[36] Staff, G. A. ; Rønquist, E. M.: Stability of the parareal algorithm. In: Kornhuber, Ralf (Hrsg.) ; et al. (Hrsg.): *Domain Decomposition Methods in Science and Engineering* Bd. 40. Berlin : Springer, 2005 (Lecture Notes in Computational Science and Engineering), 449–456

[37] Stephan, M. ; Docter, J.: JUQUEEN: IBM Blue Gene/Q® Supercomputer System at the Jülich Supercomputing Centre. In: *Journal of large-scale research facilities* 1 (2015), A1. http://dx.doi.org/10.17815/jlsrf-1-18. – DOI 10.17815/jlsrf–1–18. – ISSN 2364–091X

Jül-4392
Oktober 2015
ISSN 0944-2952